

**APPLICATION ABSTRACTION WITH DIALOG PURPOSE**

5

BACKGROUND OF THE INVENTION

The present invention generally relates to encoding computers to perform a specific application. More particularly, the present invention relates to controls for defining an application to perform  
10 recognition and/or audible prompting such as a server that generates client side markup enabled with recognition and/or audible prompting.

Small computing devices such as personal digital assistants (PDA), devices and portable phones  
15 are used with ever increasing frequency by people in their day-to-day activities. With the increase in processing power now available for microprocessors used to run these devices, the functionality of these devices are increasing, and in some cases, merging.  
20 For instance, many portable phones now can be used to access and browse the Internet as well as can be used to store personal information such as addresses, phone numbers and the like.

In view that these computing devices are  
25 being used for browsing the Internet, or are used in other server/client architectures, it is therefore necessary to enter information into the computing device. Unfortunately, due to the desire to keep these devices as small as possible in order that they  
30 are easily carried, conventional keyboards having all the letters of the alphabet as isolated buttons are usually not possible due to the limited surface area available on the housings of the computing devices.

5           To address this problem, there has been  
increased interest and adoption of using voice or  
speech to access information over a wide area network  
such as the Internet. For example, voice portals such  
as through the use of VoiceXML (voice extensible  
10 markup language) have been advanced to allow Internet  
content to be accessed using only a telephone. In  
this architecture, a document server (for example, a  
web server) processes requests from a client through  
a VoiceXML interpreter. The web server can produce  
15 VoiceXML documents in reply, which are processed by  
the VoiceXML interpreter and rendered audibly to the  
user. Using voice commands through voice recognition,  
the user can navigate the web.

          Generally, there are two techniques of  
20 "speech enabling" information or web content. In the  
first technique, existing visual markup language  
pages typically visually rendered by a device having  
a display are interpreted and rendered aurally.  
However, this approach often yields poor results  
25 because pages meant for visual interaction usually do  
not have enough information to create a sensible  
aural dialog automatically. In addition, voice  
interaction is prone to error, especially over noisy  
channels such as a telephone. Without visual or other  
30 forms of persistent feedback, navigation through the  
web server application can be extremely difficult for  
the user. This approach thus requires mechanisms such  
as help messages, which are also rendered audibly to  
the user in order to help them navigate through the

5 website. The mechanisms are commonly referred to as  
"voice dialogs", which also must address errors when  
incorrect information or no information is provided  
by the user, for example, in response to an audible  
question. Since the mechanisms are not commonly based  
10 on the visual content of the web page, they cannot be  
generated automatically, and therefore typically  
require extensive development time by the application  
developer.

A second approach to speech enabling web  
15 content, includes writing specific voice pages in a  
new language. An advantage of this approach is that  
the speech-enabled page contains all the mechanisms  
needed for aural dialog such as repairs and  
navigational help. However, a significant  
20 disadvantage is that the application pages must then  
be adapted to include the application logic as found  
in the visual content pages. In other words, the  
application logic of the visual content pages must be  
rewritten in the form of the speech-enable language.  
25 Even when this process can be automated by the use of  
tools creating visual and aural pages from the same  
specification, maintenance of the visual and speech  
enabled pages is usually difficult to synchronize. In  
addition, this approach does not easily allow  
30 multimodal applications, for example where both  
visual and speech interaction is provided on the web  
page. Since the visual and speech-enabled pages are  
unrelated, the input and output logic is not easily  
coordinated to work with each other.

5           To date, speech interaction is also  
cumbersome due to the organization or format  
currently used as the interface. Generally, the  
speech interface either tends to be tied too closely  
to the business logic of the application, which  
10 inhibits re-use of the elements of the speech  
interface in other applications, or the speech  
interface is too restricted by a simplistic dialog  
model (e.g. forms and fields).

          There is thus an ongoing need to improve  
15 upon the architecture and methods used to provide  
speech recognition in an application such as  
server/client architecture such as the Internet. In  
particular, a method, system or authoring tool that  
addresses one, several or all of the foregoing  
20 disadvantages and thus provides generation of speech-  
enabled recognition and/or speech-enabled prompting  
in an application is needed.

#### SUMMARY OF THE INVENTION

          Controls are provided for a web server to  
25 generate client side markups that include recognition  
and/or audible prompting. The controls comprise  
elements of a dialog such as a question, answer,  
confirmation, command or statement. A module forms a  
dialog by making use of the information carried in  
30 the controls.

          Each of the controls perform a role in the  
dialog. For instance, a statement control is used to  
generate corresponding markup for the client device  
to present information to the user, while a question



5 control generates markups for the client device to ask a question. An answer control generates markup for the client device so that a grammar used for recognition is associated with an input field related to a question that has been asked. If it is unclear  
10 whether or not a recognized result is correct, a confirmation control can be activated and generate markup to confirm a recognized result. A command control generates markup that allows the user to provide commands, which are other than the expected  
15 answers to a specific question, and thus, allows the user to navigate through the web server application, for example. Each of the controls includes an attribute to indicate whether the associated control is available for activation. A module, when executed such  
20 as on a client, creates a dialog to solicit and provide information as a function of which controls are activated.

#### BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a plan view of a first embodiment  
25 of a computing device operating environment.

FIG. 2 is a block diagram of the computing device of FIG. 1.

FIG. 3 is a block diagram of a general purpose computer.

30 FIG. 4 is a block diagram of an architecture for a client/server system.

FIG. 5 is a display for obtaining credit card information.

5           FIG. 6 is an exemplary page of mark-up language executable on a client having a display and voice recognition capabilities.

          FIGS. 7 is a block diagram illustrating a first approach for providing recognition and audible  
10   prompting in client side markups.

          FIG. 8 is a block diagram illustrating a second approach for providing recognition and audible prompting in client side markups.

          FIG. 9 is a block diagram illustrating a  
15   third approach for providing recognition and audible prompting in client side markups.

          FIG. 10 is a block diagram illustrating companion controls.

          FIG. 11 is a detailed block diagram  
20   illustrating companion controls.

#### DETAILED DESCRIPTION OF THE ILLUSTRATIVE EMBODIMENTS

          Before describing architecture of web based recognition and methods for implementing the same, it may be useful to describe generally computing devices  
25   that can function in the architecture. Referring now to FIG. 1, an exemplary form of a data management device (PIM, PDA or the like) is illustrated at 30. However, it is contemplated that the present invention can also be practiced using other computing  
30   devices discussed below, and in particular, those computing devices having limited surface areas for input buttons or the like. For example, phones and/or data management devices will also benefit from the present invention. Such devices will have an enhanced

5 utility compared to existing portable personal  
information management devices and other portable  
electronic devices, and the functions and compact  
size of such devices will more likely encourage the  
user to carry the device at all times. Accordingly,  
10 it is not intended that the scope of the architecture  
herein described be limited by the disclosure of an  
exemplary data management or PIM device, phone or  
computer herein illustrated.

An exemplary form of a data management  
15 mobile device 30 is illustrated in FIG. 1. The mobile  
device 30 includes a housing 32 and has an user  
interface including a display 34, which uses a  
contact sensitive display screen in conjunction with  
a stylus 33. The stylus 33 is used to press or  
20 contact the display 34 at designated coordinates to  
select a field, to selectively move a starting  
position of a cursor, or to otherwise provide command  
information such as through gestures or handwriting.  
Alternatively, or in addition, one or more buttons 35  
25 can be included on the device 30 for navigation. In  
addition, other input mechanisms such as rotatable  
wheels, rollers or the like can also be provided.  
However, it should be noted that the invention is not  
intended to be limited by these forms of input  
30 mechanisms. For instance, another form of input can  
include a visual input such as through computer  
vision.

5 Referring now to FIG. 2, a block diagram illustrates the functional components comprising the mobile device 30. A central processing unit (CPU) 50 implements the software control functions. CPU 50 is coupled to display 34 so that text and graphic icons  
10 generated in accordance with the controlling software appear on the display 34. A speaker 43 can be coupled to CPU 50 typically with a digital-to-analog converter 59 to provide an audible output. Data that is downloaded or entered by the user into the mobile  
15 device 30 is stored in a non-volatile read/write random access memory store 54 bi-directionally coupled to the CPU 50. Random access memory (RAM) 54 provides volatile storage for instructions that are executed by CPU 50, and storage for temporary data,  
20 such as register values. Default values for configuration options and other variables are stored in a read only memory (ROM) 58. ROM 58 can also be used to store the operating system software for the device that controls the basic functionality of the  
25 mobile 30 and other operating system kernel functions (e.g., the loading of software components into RAM 54).

RAM 54 also serves as a storage for the code in the manner analogous to the function of a  
30 hard drive on a PC that is used to store application programs. It should be noted that although non-volatile memory is used for storing the code, it alternatively can be stored in volatile memory that is not used for execution of the code.

5                   Wireless           signals           can           be  
transmitted/received by the mobile device through a  
wireless transceiver 52, which is coupled to CPU 50.  
An optional communication interface 60 can also be  
provided for downloading data directly from a  
10 computer (e.g., desktop computer), or from a wired  
network, if desired. Accordingly, interface 60 can  
comprise various forms of communication devices, for  
example, an infrared link, modem, a network card, or  
the like.

15                   Mobile device 30 includes a microphone 29,  
and analog-to-digital (A/D) converter 37, and an  
optional recognition program (speech, DTMF,  
handwriting, gesture or computer vision) stored in  
store 54. By way of example, in response to audible  
20 information, instructions or commands from a user of  
device 30, microphone 29 provides speech signals,  
which are digitized by A/D converter 37. The speech  
recognition program can perform normalization and/or  
feature extraction functions on the digitized speech  
25 signals to obtain intermediate speech recognition  
results. Using wireless transceiver 52 or  
communication interface 60, speech data is  
transmitted to a remote recognition server 204  
discussed below and illustrated in the architecture  
30 of FIG. 5. Recognition results are then returned to  
mobile device 30 for rendering (e.g. visual and/or  
audible) thereon, and eventual transmission to a web  
server 202 (FIG. 5), wherein the web server 202 and  
mobile device 30 operate in a client/server

5 relationship. Similar processing can be used for  
other forms of input. For example, handwriting input  
can be digitized with or without pre-processing on  
device 30. Like the speech data, this form of input  
can be transmitted to the recognition server 204 for  
10 recognition wherein the recognition results are  
returned to at least one of the device 30 and/or web  
server 202. Likewise, DTMF data, gesture data and  
visual data can be processed similarly. Depending on  
the form of input, device 30 (and the other forms of  
15 clients discussed below) would include necessary  
hardware such as a camera for visual input.

In addition to the portable or mobile  
computing devices described above, it should also be  
understood that the present invention can be used  
20 with numerous other computing devices such as a  
general desktop computer. For instance, the present  
invention will allow a user with limited physical  
abilities to input or enter text into a computer or  
other computing device when other conventional input  
25 devices, such as a full alpha-numeric keyboard, are  
too difficult to operate.

The invention is also operational with  
numerous other general purpose or special purpose  
computing systems, environments or configurations.  
30 Examples of well known computing systems,  
environments, and/or configurations that may be  
suitable for use with the invention include, but are  
not limited to, wireless or cellular telephones,  
regular telephones (without any screen), personal

5 computers, server computers, hand-held or laptop  
devices, multiprocessor systems, microprocessor-based  
systems, set top boxes, programmable consumer  
electronics, network PCs, minicomputers, mainframe  
computers, distributed computing environments that  
10 include any of the above systems or devices, and the  
like.

The following is a brief description of a  
general purpose computer 120 illustrated in FIG. 3.  
However, the computer 120 is again only one example  
15 of a suitable computing environment and is not  
intended to suggest any limitation as to the scope of  
use or functionality of the invention. Neither should  
the computer 120 be interpreted as having any  
dependency or requirement relating to any one or  
20 combination of components illustrated therein.

The invention may be described in the  
general context of computer-executable instructions,  
such as program modules, being executed by a  
computer. Generally, program modules include  
25 routines, programs, objects, components, data  
structures, etc. that perform particular tasks or  
implement particular abstract data types. The  
invention may also be practiced in distributed  
computing environments where tasks are performed by  
30 remote processing devices that are linked through a  
communications network. In a distributed computing  
environment, program modules may be located in both  
local and remote computer storage media including  
memory storage devices. Tasks performed by the

5 programs and modules are described below and with the aid of figures. Those skilled in the art can implement the description and figures as processor executable instructions, which can be written on any form of a computer readable medium.

10 With reference to FIG. 3, components of computer 120 may include, but are not limited to, a processing unit 140, a system memory 150, and a system bus 141 that couples various system components including the system memory to the processing unit  
15 140. The system bus 141 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures  
20 include Industry Standard Architecture (ISA) bus, Universal Serial Bus (USB), Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus  
25 also known as Mezzanine bus. Computer 120 typically includes a variety of computer readable mediums. Computer readable mediums can be any available media that can be accessed by computer 120 and includes both volatile and nonvolatile media, removable and  
30 non-removable media. By way of example, and not limitation, computer readable mediums may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable media



5 implemented in any method or technology for storage  
of information such as computer readable  
instructions, data structures, program modules or  
other data. Computer storage media includes, but is  
not limited to, RAM, ROM, EEPROM, flash memory or  
10 other memory technology, CD-ROM, digital versatile  
disks (DVD) or other optical disk storage, magnetic  
cassettes, magnetic tape, magnetic disk storage or  
other magnetic storage devices, or any other medium  
which can be used to store the desired information  
15 and which can be accessed by computer 120.

Communication media typically embodies  
computer readable instructions, data structures,  
program modules or other data in a modulated data  
signal such as a carrier wave or other transport  
20 mechanism and includes any information delivery  
media. The term "modulated data signal" means a  
signal that has one or more of its characteristics  
set or changed in such a manner as to encode  
information in the signal. By way of example, and  
25 not limitation, communication media includes wired  
media such as a wired network or direct-wired  
connection, and wireless media such as acoustic, FR,  
infrared and other wireless media. Combinations of  
any of the above should also be included within the  
30 scope of computer readable media.

The system memory 150 includes computer  
storage media in the form of volatile and/or  
nonvolatile memory such as read only memory (ROM) 151  
and random access memory (RAM) 152. A basic

5 input/output system 153 (BIOS), containing the basic routines that help to transfer information between elements within computer 120, such as during start-up, is typically stored in ROM 151. RAM 152 typically contains data and/or program modules that  
10 are immediately accessible to and/or presently being operated on by processing unit 140. By way of example, and not limitation, FIG. 3 illustrates operating system 54, application programs 155, other program modules 156, and program data 157.

15 The computer 120 may also include other removable/non-removable volatile/nonvolatile computer storage media. By way of example only, FIG. 3 illustrates a hard disk drive 161 that reads from or writes to non-removable, nonvolatile magnetic media,  
20 a magnetic disk drive 171 that reads from or writes to a removable, nonvolatile magnetic disk 172, and an optical disk drive 175 that reads from or writes to a removable, nonvolatile optical disk 176 such as a CD ROM or other optical media. Other removable/non-  
25 removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid  
30 state ROM, and the like. The hard disk drive 161 is typically connected to the system bus 141 through a non-removable memory interface such as interface 160, and magnetic disk drive 171 and optical disk drive

5 175 are typically connected to the system bus 141 by  
a removable memory interface, such as interface 170.

The drives and their associated computer  
storage media discussed above and illustrated in FIG.  
3, provide storage of computer readable instructions,  
10 data structures, program modules and other data for  
the computer 120. In FIG. 3, for example, hard disk  
drive 161 is illustrated as storing operating system  
164, application programs 165, other program modules  
166, and program data 167. Note that these components  
15 can either be the same as or different from operating  
system 154, application programs 155, other program  
modules 156, and program data 157. Operating system  
164, application programs 165, other program modules  
166, and program data 167 are given different numbers  
20 here to illustrate that, at a minimum, they are  
different copies.

A user may enter commands and information  
into the computer 120 through input devices such as a  
keyboard 182, a microphone 183, and a pointing device  
25 181, such as a mouse, trackball or touch pad. Other  
input devices (not shown) may include a joystick,  
game pad, satellite dish, scanner, or the like.  
These and other input devices are often connected to  
the processing unit 140 through a user input  
30 interface 180 that is coupled to the system bus, but  
may be connected by other interface and bus  
structures, such as a parallel port, game port or a  
universal serial bus (USB). A monitor 184 or other  
type of display device is also connected to the

5 system bus 141 via an interface, such as a video interface 185. In addition to the monitor, computers may also include other peripheral output devices such as speakers 187 and printer 186, which may be connected through an output peripheral interface 188.

10 The computer 120 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 194. The remote computer 194 may be a personal computer, a hand-held device, a server, a router, a network PC, a  
15 peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 120. The logical connections depicted in FIG. 3 include a local area network (LAN) 191 and a wide area network (WAN) 193, but may also include other networks. Such  
20 networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment,  
25 the computer 120 is connected to the LAN 191 through a network interface or adapter 190. When used in a WAN networking environment, the computer 120 typically includes a modem 192 or other means for establishing communications over the WAN 193, such as  
30 the Internet. The modem 192, which may be internal or external, may be connected to the system bus 141 via the user input interface 180, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 120, or

5 portions thereof, may be stored in the remote memory  
storage device. By way of example, and not  
limitation, FIG. 3 illustrates remote application  
programs 195 as residing on remote computer 194. It  
will be appreciated that the network connections  
10 shown are exemplary and other means of establishing a  
communications link between the computers may be  
used.

#### EXEMPLARY ARCHITECTURE

FIG. 4 illustrates architecture 200 for web  
15 based recognition as can be used with the present  
invention. Generally, information stored in a web  
server 202 can be accessed through mobile device 30  
(which herein also represents other forms of  
computing devices having a display screen, a  
20 microphone, a camera, a touch sensitive panel, etc.,  
as required based on the form of input), or through  
phone 80 wherein information is requested audibly or  
through tones generated by phone 80 in response to  
keys depressed and wherein information from web  
25 server 202 is provided only audibly back to the user.

In this exemplary embodiment, Architecture  
200 is unified in that whether information is  
obtained through device 30 or phone 80 using speech  
recognition, a single recognition server 204 can  
30 support either mode of operation. In addition,  
architecture 200 operates using an extension of well-  
known markup languages (e.g. HTML, XHTML, CHTML, XML,  
WML, and the like). Thus, information stored on web  
server 202 can also be accessed using well-known GUI

5 methods found in these markup languages. By using an  
extension of well-known markup languages, authoring  
on the web server 202 is easier, and legacy  
applications currently existing can be also easily  
modified to include voice or other forms of  
10 recognition.

Generally, device 30 executes HTML+  
scripts, or the like, provided by web server 202.  
When voice recognition is required, by way of  
example, speech data, which can be digitized audio  
15 signals or speech features wherein the audio signals  
have been preprocessed by device 30 as discussed  
above, are provided to recognition server 204 with an  
indication of a grammar or language model to use  
during speech recognition. The implementation of the  
20 recognition server 204 can take many forms, one of  
which is illustrated, but generally includes a  
recognizer 211. The results of recognition are  
provided back to device 30 for local rendering if  
desired or appropriate. Upon compilation of  
25 information through recognition and any graphical  
user interface if used, device 30 sends the  
information to web server 202 for further processing  
and receipt of further HTML scripts, if necessary.

As illustrated in FIG. 4, device 30, web  
30 server 202 and recognition server 204 are commonly  
connected, and separately addressable, through a  
network 205, herein a wide area network such as the  
Internet. It therefore is not necessary that any of  
these devices be physically located adjacent to each

5 other. In particular, it is not necessary that web  
server 202 includes recognition server 204. In this  
manner, authoring at web server 202 can be focused on  
the application to which it is intended without the  
authors needing to know the intricacies of  
10 recognition server 204. Rather, recognition server  
204 can be independently designed and connected to  
the network 205, and thereby, be updated and improved  
without further changes required at web server 202.  
As discussed below, web server 202 can also include  
15 an authoring mechanism that can dynamically generate  
client-side markups and scripts. In a further  
embodiment, the web server 202, recognition server  
204 and client 30 may be combined depending on the  
capabilities of the implementing machines. For  
20 instance, if the client comprises a general purpose  
computer, e.g. a personal computer, the client may  
include the recognition server 204. Likewise, if  
desired, the web server 202 and recognition server  
204 can be incorporated into a single machine.

25 Access to web server 202 through phone 80  
includes connection of phone 80 to a wired or  
wireless telephone network 208, that in turn,  
connects phone 80 to a third party gateway 210.  
Gateway 210 connects phone 80 to a telephony voice  
30 browser 212. Telephone voice browser 212 includes a  
media server 214 that provides a telephony interface  
and a voice browser 216. Like device 30, telephony  
voice browser 212 receives HTML scripts or the like  
from web server 202. In one embodiment, the HTML

5 scripts are of the form similar to HTML scripts  
provided to device 30. In this manner, web server 202  
need not support device 30 and phone 80 separately,  
or even support standard GUI clients separately.  
Rather, a common markup language can be used. In  
10 addition, like device 30, voice recognition from  
audible signals transmitted by phone 80 are provided  
from voice browser 216 to recognition server 204,  
either through the network 205, or through a  
dedicated line 207, for example, using TCP/IP. Web  
15 server 202, recognition server 204 and telephone  
voice browser 212 can be embodied in any suitable  
computing environment such as the general purpose  
desktop computer illustrated in FIG. 3.

However, it should be noted that if DTMF  
20 recognition is employed, this form of recognition  
would generally be performed at the media server 214,  
rather than at the recognition server 204. In other  
words, the DTMF grammar would be used by the media  
server 214.

25 Referring back to FIG. 4, web server 202  
can include a server side plug-in authoring tool or  
module 209 (e.g. ASP, ASP+, ASP.Net by Microsoft  
Corporation, JSP, Javabeans, or the like). Server  
side plug-in module 209 can dynamically generate  
30 client-side markups and even a specific form of  
markup for the type of client accessing the web  
server 202. The client information can be provided to  
the web server 202 upon initial establishment of the  
client/server relationship, or the web server 202 can



5 include modules or routines to detect the capabilities of the client device. In this manner, server side plug-in module 209 can generate a client side markup for each of the voice recognition scenarios, i.e. voice only through phone 80 or  
10 multimodal for device 30. By using a consistent client side model, application authoring for many different clients is significantly easier.

In addition to dynamically generating client side markups, high-level dialog modules, discussed below, can be implemented as a server-side control stored in store 211 for use by developers in application authoring. In general, the high-level dialog modules 211 would generate dynamically client-side markup and script in both voice-only and  
15 multimodal scenarios based on parameters specified by developers. The high-level dialog modules 211 can include parameters to generate client-side markups to fit the developers' needs.  
20

#### EXEMPLARY CLIENT SIDE EXTENSIONS

25 Before describing dynamic generation of client-side markups to which the present invention is directed, it may be helpful to first discuss an exemplary form of extensions to the markup language for use in web based recognition.

30 As indicated above, the markup languages such as HTML, XHTML cHTML, XML, WML or any other SGML-derived markup, which are used for interaction between the web server 202 and the client device 30, are extended to include controls and/or objects that

5 provide recognition in a client/server architecture. Generally, controls and/or objects can include one or more of the following functions: recognizer controls and/or objects for recognizer configuration, recognizer execution and/or post-processing;

10 synthesizer controls and/or objects for synthesizer configuration and prompt playing; grammar controls and/or objects for specifying input grammar resources; and/or binding controls and/or objects for processing recognition results. The extensions are

15 designed to be a lightweight markup layer, which adds the power of an audible, visual, handwriting, etc. interface to existing markup languages. As such, the extensions can remain independent of: the high-level page in which they are contained, e.g. HTML; the low-

20 level formats which the extensions used to refer to linguistic resources, e.g. the text-to-speech and grammar formats; and the individual properties of the recognition and speech synthesis platforms used in the recognition server 204. Although speech

25 recognition will be discussed below, it should be understood that the techniques, tags and server side controls described hereinafter can be similarly applied in handwriting recognition, gesture recognition and image recognition.

30 In the exemplary embodiment, the extensions (also commonly known as "tags") are a small set of XML elements, with associated attributes and DOM object properties, events and methods, which may be used in conjunction with a source markup document to

5    apply a recognition and/or audible prompting  
interface, DTMF or call control to a source page. The  
extensions' formalities and semantics are independent  
of the nature of the source document, so the  
extensions can be used equally effectively within  
10 HTML, XHTML, cHTML, XML, WML, or with any other SGML-  
derived markup. The extensions follow the document  
object model wherein new functional objects or  
elements, which can be hierarchical, are provided.  
Each of the elements are discussed in detail in the  
15 Appendix, but generally the elements can include  
attributes, properties, methods, events and/or other  
"child" elements.

At this point, it should also be noted that  
the extensions may be interpreted in two different  
20 "modes" according to the capabilities of the device  
upon which the browser is being executed on. In a  
first mode, "object mode", the full capabilities are  
available. The programmatic manipulation of the  
extensions by an application is performed by whatever  
25 mechanisms are enabled by the browser on the device,  
e.g. a JScript interpreter in an XHTML browser, or a  
WMLScript interpreter in a WML browser. For this  
reason, only a small set of core properties and  
methods of the extensions need to be defined, and  
30 these manipulated by whatever programmatic mechanisms  
exist on the device or client side. The object mode  
provides eventing and scripting and can offer greater  
functionality to give the dialog author a much finer  
client-side control over speech interactions. As used

5 herein, a browser that supports full event and  
scripting is called an "uplevel browser". This form  
of a browser will support all the attributes,  
properties, methods and events of the extensions.  
Uplevel browsers are commonly found on devices with  
10 greater processing capabilities.

The extensions can also be supported in a  
"declarative mode". As used herein, a browser  
operating in a declarative mode is called a  
"downlevel browser" and does not support full  
15 eventing and scripting capabilities. Rather, this  
form of browser will support the declarative aspects  
of a given extension (i.e. the core element and  
attributes), but not all the DOM (document object  
model) object properties, methods and events. This  
20 mode employs exclusively declarative syntax, and may  
further be used in conjunction with declarative  
multimedia synchronization and coordination  
mechanisms (synchronized markup language) such as  
SMIL (Synchronized Multimedia Integration Language)  
25 2.0. Downlevel browsers will typically be found on  
devices with limited processing capabilities.

At this point though, a particular mode of  
entry should be discussed. In particular, use of  
speech recognition in conjunction with at least a  
30 display and, in a further embodiment, a pointing  
device as well which enables the coordination of  
multiple modes of input, e.g. to indicate the fields  
for data entry, is particularly useful. Specifically,  
in this mode of data entry, the user is generally

5 able to coordinate the actions of the pointing device  
with the speech input, so for example the user is  
under control of when to select a field and provide  
corresponding information relevant to the field. For  
instance, a credit card submission graphical user  
10 interface (GUI) is illustrated in FIG. 5, a user  
could first decide to enter the credit card number in  
field 252 and then enter the type of credit card in  
field 250 followed by the expiration date in field  
254. Likewise, the user could return back to field  
15 252 and correct an errant entry, if desired. When  
combined with speech recognition, an easy and natural  
form of navigation is provided. As used herein, this  
form of entry using both a screen display allowing  
free form actions of the pointing device on the  
20 screen, e.g. the selection of fields and recognition  
is called "multimodal".

Referring to FIG. 6, a HTML markup language  
code is illustrated. The HTML code includes a body  
portion 270 and a script portion 272. Entry of  
25 information in each of the fields 250, 252 and 254 is  
controlled or executed by code portions 280, 282 and  
284, respectively. Referring first to code portion  
280, on selection of field 250, for example, by use  
of stylus 33 of device 30, the event "onClick" is  
30 initiated which calls or executes function "talk" in  
script portion 272. This action activates a grammar  
used for speech recognition that is associated with  
the type of data generally expected in field 250.  
This type of interaction, which involves more than

5 one technique of input (e.g. voice and pen-click/roller) is referred as "multimodal".

Referring now back to the grammar, the grammar is a syntactic grammar such as but not limited to a context-free grammar, a N-grammar or a  
10 hybrid grammar. (Of course, DTMF grammars, handwriting grammars, gesture grammars and image grammars would be used when corresponding forms of recognition are employed. As used herein, a "grammar" includes information for performing recognition, and  
15 in a further embodiment, information corresponding to expected input to be entered, for example, in a specific field.) A control 290 (herein identified as "reco") includes various elements, two of which are illustrated, namely a grammar element "grammar" and a  
20 "bind" element. Generally, like the code downloaded to a client from web server 202, the grammars can originate at web server 202 and be downloaded to the client and/or forwarded to a remote server for speech processing. The grammars can then be stored locally  
25 thereon in a cache. Eventually, the grammars are provided to the recognition server 204 for use in recognition. The grammar element is used to specify grammars, either inline or referenced using an attribute.

30 Upon receipt of recognition results from recognition server 204 corresponding to the recognized speech, handwriting, gesture, image, etc., syntax of reco control 290 is provided to receive the corresponding results and associate it with the

5 corresponding field, which can include rendering of  
the text therein on display 34. In the illustrated  
embodiment, upon completion of speech recognition  
with the result sent back to the client, it  
deactivates the reco object and associates the  
10 recognized text with the corresponding field.  
Portions 282 and 284 operate similarly wherein unique  
reco objects and grammars are called for each of the  
fields 252 and 254 and upon receipt of the recognized  
text is associated with each of the fields 252 and  
15 254. With respect to receipt of the card number field  
252, the function "handle" checks the length of the  
card number with respect to the card type.

#### GENERATION OF CLIENT SIDE MARKUPS

As indicated above, server side plug-in  
20 module 209 outputs client side markups when a request  
has been made from the client device 30. In short,  
the server side plug-in module 209 allows the  
website, and thus, the application and services  
provided by the application to be defined or  
25 constructed. The instructions in the server side  
plug-in module 209 are made of a complied code. The  
code is run when a web request reaches the web server  
202. The server side plug-in module 209 then outputs  
a new client side markup page that is sent to the  
30 client device 30. As is well known, this process is  
commonly referred to as rendering. The server side  
plug-in module 209 operates on "controls" that  
abstract and encapsulate the markup language, and  
thus, the code of the client side markup page. Such

5 controls that abstract and encapsulate the markup language and operate on the webserver 202 include or are equivalent to "Servlets" or "Server-side plug ins" to name a few.

As is known, server side plug-in modules of  
10 the prior art can generate client side markup for visual rendering and interaction with the client device 30. Three different approaches are provided herein for extending the server side plug-in module 209 to include recognition and audible prompting  
15 extensions such as the exemplary client side extensions discussed above. In a first approach illustrated schematically in Fig. 7, the current, visual, server side controls (which include parameters for visual display such as location for  
20 rendering, font, foreground color, background color, etc.) are extended to include parameters or attributes for recognition and audibly prompting for related recognition. Using speech recognition and associated audible prompting by way of example, the  
25 attributes generally pertain to audible prompting parameters such as whether the prompt comprises inline text for text-to-speech conversion, playing of a prerecorded audio file (e.g. a wave file), the location of the data (text for text-to-speech  
30 conversion or a prerecorded audio file) for audible rendering, etc. For recognition, the parameters or attributes can include the location of the grammar to be used during recognition, confidence level thresholds, etc. Since the server side plug-in module



5 209 generates client side markup, the parameters and attributes for the controls for the server side plug-in module 209 relate to the extensions provided in the client side markup for recognition and/or audible prompting.

10           The controls indicated at 300A in Fig. 7 are controls, which are well-known in website application development or authoring tools such as ASP, ASP+, ASP.Net, JSP, Javabeans, or the like. Such controls are commonly formed in a library and used by  
15 controls 302 to perform a particular visual task. Library 300A includes methods for generating the desired client markup, event handlers, etc. Examples of visual controls 302 include a "Label" control that provides a selected text label on a visual display  
20 such as the label "Credit Card Submission" 304 in Fig. 5. Another example of a higher level visual control 302 is a "Textbox", which allows data to be entered in a data field such as is indicated at 250 in Fig. 5. The existing visual controls 302 are also  
25 well-known. In the first approach for extending server side plug-in module controls to include recognition and/or audible prompting, each of the visual controls 302 would include further parameters or attributes related to recognition or audible  
30 prompting. In the case of the "label" control, which otherwise provides selected text on a visual display, further attributes may include whether an audio data file will be rendered or text-to-speech conversion will be employed as well as the location of this data

5 file. A library 300B, similar to library 300A,  
includes further markup information for performing  
recognition and/or audible prompting. Each of the  
visual controls 302 is coded so as to provide this  
information to the controls 300B as appropriate to  
10 perform the particular task related to recognition or  
audible prompting.

As another example, the "Textbox" control,  
which generates an input field on a visual display  
and allows the user of the client device 30 to enter  
15 information, would also include appropriate  
recognition or audible prompting parameters or  
attributes such as the grammar to be used for  
recognition. It should be noted that the recognition  
or audible prompting parameters are optional and need  
20 not be used if recognition or audible prompting is  
not otherwise desired.

In general, if a control at level 302  
includes parameters that pertain to visual aspects,  
the control will access and use the library 300A.  
25 Likewise, if the control includes parameters  
pertaining to recognition and/or audible prompting  
the control will access or use the library 300B. It  
should be noted that libraries 300A and 300B have  
been illustrated separately in order to emphasize the  
30 additional information present in library 300B and  
that a single library having the information of  
libraries 300A and 300B can be implemented.

5           In this approach, each of the current or  
prior art visual controls 302 are extended to include  
appropriate recognition/audible prompting attributes.  
The controls 302 can be formed in a library. The  
server side plug-in module 209 accesses the library  
10 for markup information. Execution of the controls  
generates a client side markup page, or a portion  
thereof, with the provided parameters.

          In a second approach illustrated in Fig. 8,  
new visual, recognition/audible prompting controls  
15 304 are provided such that the controls 304 are a  
subclass relative to visual controls 302, wherein  
recognition/audible prompting functionality or markup  
information is provided at controls 304. In other  
words, a new set of controls 304 are provided for  
20 recognition/audible prompting and include appropriate  
parameters or attributes to perform the desired  
recognition or an audible prompting related to a  
recognition task on the client device 30. The  
controls 304 use the existing visual controls 302 to  
25 the extent that visual information is rendered or  
obtained through a display. For instance, a control  
"SpeechLabel" at level 304 uses the "Label" control  
at level 302 to provide an audible rendering and/or  
visual text rendering. Likewise, a "SpeechTextbox"  
30 control would associate a grammar and related  
recognition resources and processing with an input  
field. Like the first approach, the attributes for  
controls 304 include where the grammar is located for  
recognition, the inline text for text-to-speech

5 conversion, or the location of a prerecorded audio  
data file that will be rendered directly or a text  
file through text-to-speech conversion. The second  
approach is advantageous in that interactions of the  
recognition controls 304 with the visual controls 302  
10 are through parameters or attributes, and thus,  
changes in the visual controls 302 may not require  
any changes in the recognition controls 304 provided  
the parameters or attributes interfacing between the  
controls 304 and 302 are still appropriate. However,  
15 with the creation of further visual controls 302, a  
corresponding recognition/audible prompting control  
at level 304 may also have to be written.

A third approach is illustrated in Fig. 9.  
Generally, controls 306 of the third approach are  
20 separate from the visual controls 302, but are  
associated selectively therewith as discussed below.  
In this manner, the controls 306 do not directly  
build upon the visual controls 302, but rather  
provide recognition/audible prompting enablement  
25 without having to rewrite the visual controls 302.  
The controls 306, like the controls 302, use a  
library 300. In this embodiment, library 300 includes  
both visual and recognition/audible prompting markup  
information and as such is a combination of libraries  
30 300A and 300B of Fig. 7.

There are significant advantages to this  
third approach. Firstly, the visual controls 302 do  
not need to be changed in content. Secondly, the  
controls 306 can form a single module which is

5 consistent and does not need to change according to  
the nature of the speech-enabled control 302.  
Thirdly, the process of speech enablement, that is,  
the explicit association of the controls 306 with the  
visual controls 302 is fully under the developer's  
10 control at design time, since it is an explicit and  
selective process. This also makes it possible for  
the markup language of the visual controls to receive  
input values from multiple sources such as through  
recognition provided by the markup language generated  
15 by controls 306, or through a conventional input  
device such as a keyboard. In short, the controls 306  
can be added to an existing application authoring  
page of a visual authoring page of the server side  
plug-in module 209. The controls 306 provide a new  
20 modality of interaction (i.e. recognition and/or  
audible prompting) for the user of the client device  
30, while reusing the visual controls' application  
logic and visual input/output capabilities. In view  
that the controls 306 can be associated with the  
25 visual controls 302 whereat the application logic can  
be coded, controls 306 may be hereinafter referred to  
as "companion controls 306" and the visual controls  
302 be referred to as "primary controls 302". It  
should be noted that these references are provided  
30 for purposes of distinguishing controls 302 and 306  
and are not intended to be limiting. For instance,  
the companion controls 306 could be used to develop  
or author a website that does not include visual  
renderings such as a voice-only website. In such a

5 case, certain application logic could be embodied in the companion control logic.

An exemplary set of companion controls 306 are further illustrated in Fig. 10. The set of companion controls 306 can be grouped as output  
10 controls 308 and input controls 310. Output controls 308 provide "prompting" client side markups, which typically involves the playing of a prerecorded audio file, or text for text-to-speech conversion, the data included in the markup directly or referenced via a  
15 URL. Although a single output control can be defined with parameters to handle all audible prompting, and thus should be considered as a further aspect of the present invention, in the exemplary embodiment, the forms or types of audible prompting in a human dialog are formed as separate controls. In particular, the  
20 output controls 308 can include a "Question" control 308A, a "Confirmation" control 308B and a "Statement" control 308C, which will be discussed in detail below. Likewise, the input controls 310 can also form  
25 or follow human dialog and include a "Answer" control 310A and a "Command" control 310B. The input controls 310 are discussed below, but generally the input controls 310 associate a grammar with expected or possible input from the user of the client device 30.

30 Although the question control 308A, confirmation control 308B, statement control 308C, answer control 310A, command control 310B, other controls as well as the general structure of these controls, the parameters and event handlers, are

5 specifically discussed with respect to use as  
companion controls 306, it should be understood that  
these controls, the general structure, parameters and  
event handlers can be adapted to provide recognition  
and/or audible prompting in the other two approaches  
10 discussed above with respect to Figs. 7 and 8. For  
instance, the parameter "ClientToSpeechEnable", which  
comprises one exemplary mechanism to form the  
association between a companion control and a visual  
control, would not be needed when embodied in the  
15 approaches of Figs. 7 and 8.

In a multimodal application, at least one  
of the output controls 308 or one of the input  
controls 310 is associated with a primary or visual  
control 302. In the embodiment illustrated, the  
20 output controls 308 and input controls 310 are  
arranged or organized under a "Question/Answer"  
(hereinafter also "QA") control 320. QA control 320  
is executed on the web server 202, which means it is  
defined on the application development web page held  
25 on the web server using the server-side markup  
formalism (ASP, JSP or the like), but is output as a  
different form of markup to the client device 30.  
Although illustrated in Fig. 10 where the QA control  
appears to be formed of all of the output controls  
30 308 and the input controls 310, it should be  
understood that these are merely options wherein one  
or more may be included for a QA control.

5           At this point it may be helpful to explain  
use of the controls 308 and 310 in terms of  
application scenarios. Referring to Fig. 11 and in a  
voice-only application QA control 320 could comprise  
a single question control 308A and an answer control  
10 310A. The question control 308A contains one or more  
prompt objects or controls 322, while the answer  
control 310A can define a grammar through grammar  
object or control 324 for recognition of the input  
data and related processing on that input. Line 326  
15 represents the association of the QA control 320 with  
the corresponding primary control 302, if used. In a  
multimodal scenario, where the user of the client  
device 30 may touch on the visual textbox, for  
example with a "TapEvent", an audible prompt may not  
20 be necessary. For example, for a primary control  
comprising a textbox having visual text forming an  
indication of what the user of client device should  
enter in the corresponding field, a corresponding QA  
control 320 may or may not have a corresponding  
25 prompt such as an audio playback or a text-to-speech  
conversion, but would have a grammar corresponding to  
the expected value for recognition, and event  
handlers 328 to process the input, or process other  
recognizer events such as no speech detected, speech  
30 not recognized, or events fired on timeouts (as  
illustrated in "Eventing" below).

In general, the QA control through the  
output controls 308 and input controls 310 and  
additional logic can perform one or more of the



5 following: provide output audible prompting, collect  
input data, perform confidence validation of the  
input result, allow additional types of input such as  
"help" commands, or commands that allow the user of  
the client device to navigate to other selected areas  
10 of the website, allow confirmation of input data and  
control of dialog flow at the website, to name a few.  
In short, the QA control 320 contains all the  
controls related to a specific topic. In this manner,  
a dialog is created through use of the controls with  
15 respect to the topic in order to inform to obtain  
information, to confirm validity, or to repair a  
dialog or change the topic of conversation.

In one method of development, the  
application developer can define the visual layout of  
20 the application using the visual controls 302. The  
application developer can then define the spoken  
interface of the application using companion controls  
306 (embodied as QA control 320, or output controls  
308 and input control 310). As illustrated in FIGS.  
25 10 and 11, each of the companion controls 306 are  
then linked or otherwise associated with the  
corresponding primary or visual control 302 to  
provide recognition and audible prompting. Of course  
if desired, the application developer can define or  
30 encode the application by switching between visual  
controls 302 and companion controls 306, forming the  
links therebetween, until the application is  
completely defined or encoded.

5           At this point, it may be helpful to provide  
a short description of each of the output controls  
308 and input controls 310. Detailed descriptions are  
provided below in Appendix B.

#### Questions, Answers and Commands

10           Generally, as indicated above, the question  
controls 308A and answer controls 310A in a QA  
control 320 hold the prompt and grammar resources  
relevant to the primary control 302, and related  
binding (associating recognition results with input  
15 fields of the client-side markup page) and processing  
logic. The presence, or not, of question controls  
308A and answer controls 310A determines whether  
speech output or recognition input is enabled on  
activation. Command controls 310B and user initiative  
20 answers are activated by specification of the Scope  
property on the answer controls 310A and command  
controls 310B.

In simple voice-only applications, a QA  
control 320 will typically hold one question control  
25 or object 308A and one answer control or object 310A.  
Although not shown in the example below, command  
controls 310B may also be specified, e.g. Help,  
Repeat, Cancel, etc., to enable user input which does  
not directly relate to the answering of a particular  
30 question.

A typical 'regular' QA control for voice-only  
dialog is as follows:

<Speech:QA

```

5         id="QA_WhichOne"
          ControlsToSpeechEnable="textBox1"
          runat="server" >

          <Question >
10         <prompt> Which one do you want?
          </prompt>
          </Question>
          <Answer >
          <grammar src="whichOne.gram" />
15         </Answer>
          </Speech:QA>

```

(The examples provided herein are written in the ASP.Net framework by example only and should not be considered as limiting the present invention.)

In this example, the QA control can be identified by its "id", while the association of the QA control with the desired primary or visual control is obtained through the parameter "ControlsToSpeechEnable", which identifies one or more primary controls by their respective identifiers. If desired, other well-known techniques can be used to form the association. For instance, direct, implicit associations are available through the first and second approaches described above, or separate tables can be created used to maintain the associations. The parameter "runat" instructs the web server that this code should be executed at the webserver 202 to generate the correct markup.

A QA control might also hold only a statement control 308C, in which case it is a prompt-only control without active grammars (e.g. for a welcome prompt). Similarly a QA control might hold

5 only an answer control 310A, in which case it may be a multimodal control, whose answer control 310A activates its grammars directly as the result of an event from the GUI, or a scoped mechanism (discussed below) for user initiative.

10 It should also be noted that a QA control 320 may also hold multiple output controls 308 and input controls 310 such as multiple question controls 308A and multiple answers controls 310A. This allows an author to describe interactional flow about the  
 15 same entity within the same QA control. This is particularly useful for more complex voice-only dialogs. So a mini-dialog which may involve different kinds of question and answer (e.g. asking, confirming, giving help, etc.), can be specified  
 20 within the wrapper of the QA control associated with the visual control which represents the dialog entity. A complex QA control is illustrated in Fig. 11.

The foregoing represent the main features  
 25 of the QA control. Each feature is described from a functional perspective below.

#### Answer Control

The answer control 310A abstracts the  
 notion of grammars, binding and other recognition  
 30 processing into a single object or control. Answer controls 310A can be used to specify a set of possible grammars relevant to a question, along with binding declarations and relevant scripts. Answer

5 controls for multimodal applications such as "Tap-  
and-Talk" are activated and deactivated by GUI  
browser events. The following example illustrates an  
answer control 310A used in a multimodal application  
to select a departure city on the "mouseDown" event  
10 of the textbox "txtDepCity", and write its value into  
the primary textbox control:

```

20      <Speech:QA
          controlsToSpeechEnable="txtDepCity"
15      runat="server">
          <Answer      id="AnsDepCity"
              StartEvent="onMouseDown"
              StopEvent="onMouseUp"
              />
          <grammar src="/grammars/depCities.gram"/>
          <bind value="//sml/DepCity"
20      targetElement="txtCity" />
          </Answer>
      </Speech:QA>
25
      Typical answer controls 310A in voice-only
      applications are activated directly by question
      controls 308A as described below.

```

30 The answer control further includes a  
mechanism to associate a received result with the  
primary controls. Herein, binding places the values  
in the primary controls; however, in another  
embodiment the association mechanism may allow the  
primary control to look at or otherwise access the  
35 recognized results.

#### Question Control

Question controls 308A abstracts the notion  
of the prompt tags (Appendix A) into an object which

5 contains a selection of possible prompts and the  
 answer controls 310A which are considered responses  
 to the question. Each question control 308A is able  
 to specify which answer control 310A it activates on  
 its execution. This permits appropriate response  
 10 grammars to be bundled into answer controls 310A,  
 which reflect relevant question controls 308A.

The following question control 308A might  
 be used in a voice-only application to ask for a  
 Departure City:

```

15      <Speech:QA id="QADepCity"
           controlsToSpeechEnable="txtDepCity"
           runat="server" >
           <Question id="Q1" Answers="AnsDepCity" >
20             <prompt>
                 Please give me the departure
                 city.
             </prompt>
           </Question>
25
           <Answer id="AnsDepCity" ... />
        </Speech:QA>
  
```

In the example below, different prompts can  
 30 be called depending on an internal condition of the  
 question control 308A. The ability to specify  
 conditional tests on the prompts inside a question  
 control 308A means that changes in wording can be  
 accommodated within the same functional unit of the  
 35 question control 308A.

```

        <Speech:QA      id="QADepCity"
           controlsToSpeechEnable="txtDepCity"
           runat="server" >
40      <Question id="Q1" Answers="AnsDepCity" >
  
```

```

5      <prompt count="1">
        Now I need to get the departure city.
        Where would you like to fly from?
      </prompt>
      <prompt count="2">
10      Which departure city?
      </prompt>
      </Question>

      <Answer id="AnsDepCity" ... />
15 </Speech:QA>

```

#### Conditional QA Control

The following example illustrates how to determine whether or not to activate a QA control based upon information known to the application. The example is a portion of a survey application. The survey is gathering information from employees regarding the mode of transportation they use to get to work.

The portion of the survey first asks whether or not the user rides the bus to work. If the answer is:

- Yes, the next question asks how many days last week the users rode the bus.
- No, the "number of days rode the bus" question is bypassed.

```

35 <asp:Label id="lblDisplay1"
      text="Do you ride the bus to work?"
      runat="server"/>

      <asp:DropDownList id="lstRodeBusYN" runat="server">
        <asp:ListItem
40 selected="true">No</asp:ListItem>
        <asp:ListItem>Yes</asp:ListItem>
      </asp:DropDownList>

```

```

5  </asp:DropDownList>

    <Speech:QA id="QA_RideBus
        ControlsToSpeechEnable="lstRodeBusYN"
        runat="server" >
10
        <SDN:Question id="Q_RideBus" >

            <prompt bargeIn="False">
                Do you ride the bus to work?
15            </prompt>

        </SDN:Question>

        <SDN:Answer id="A_RideBus"
20            autobind="False"
            StartEvent="onMouseDown"
            StopEvent="onMouseUp"
            runat="server"

25            onClientReco="ProcessRideBusAnswer"
            <grammar src="..." />    <--! "yes/no"
            grammar -->

        </SDN:Answer>
30 </Speech:QA>

    <asp:Label id="lblDisplay2"
        enabled="False"
        text="How many days last week did you ride
35    the bus to work?"
        runat="server"/>

    <asp:DropDownList id="lstDaysRodeBus" enabled="False"
        runat="server">
40        <asp:ListItem selected="true"
        >0</asp:ListItem>
        <asp:ListItem>1</asp:ListItem>
        <asp:ListItem>2</asp:ListItem>
        <asp:ListItem>3</asp:ListItem>
45        <asp:ListItem>4</asp:ListItem>
        <asp:ListItem>5</asp:ListItem>
        <asp:ListItem>6</asp:ListItem>
        <asp:ListItem>7</asp:ListItem>

```



```

5          </asp:DropDownList>

<Speech:QA id="QA_DaysRodeBus"
           ControlsToSpeechEnable="lstDaysRodeBus"
           ClientTest="RideBusCheck"
10           runat="server" >
           <Question id="Q_DaysRodeBus" >

               <prompt bargeIn="False">
                   How many days last week did you ride the
15 bus to work?
               </prompt>

               </SDN:Question>

20           <SDN:Answer id="A_DaysRodeBus"
                       autobind="False"
                       StartEvent="onMouseDown"
                       StopEvent="onMouseUp"
                       runat="server"

25           onClientReco="ProcessDaysRodeBusAnswer"
               <grammar src="..." />   <--! "numbers"
grammar -->

30           </SDN:Answer>

</Speech:QA>

35 <script language="jscript">
    function ProcessRideBusAnswer() {
        <--! using SML attribute of the Event object,
        determine yes or no answer -->
40        <--! then select the appropriate item in the
        dropdown listbox -->
        <--! and enable the next label and dropdown
        listbox if answer is "yes" -->

45        if <--! Answer is "yes" --> {
            lstRodeBusYN.selectedIndex=2
            lblDisplay2.enabled="true"
            lstDaysRodeBus.enabled="true" }

```

```

5      }

      function RideBusCheck() {
          if lstRodeBusYN.selectedIndex="1"  <--!
this is no -->
10          then return "False"
          endif
      }

      function ProcessDaysRodeBusAnswer() {
15          <--! case statement to select proper
dropdown item -->

      }
</script>
20

```

In the example provided above, the QA control "QA\_DaysRodeBus" is executed based on a boolean parameter "ClientTest", which in this example, is set based on the function RideBusCheck().

If the function returns a false condition, the QA control is not activated, whereas if a true condition is returned the QA control is activated. The use of an activation mechanism allows increased flexibility and improved dialog flow in the client side markup page produced. As indicated in Appendix B many of the controls and objects include an activation mechanism.

#### Command Control

Command controls 310B are user utterances common in voice-only dialogs which typically have little semantic import in terms of the question asked, but rather seek assistance or effect navigation, e.g. help, cancel, repeat, etc. The Command control 310B within a QA control 306 can be

5 used to specify not only the grammar and associated processing on recognition (rather like an answer control 310A without binding of the result to an input field), but also a 'scope' of context and a type. This allows for the authoring of both global  
 10 and context-sensitive behavior on the client side markup.

As appreciated by those skilled in the art from the foregoing description, controls 306 can be organized in a tree structure similar to that used in  
 15 visual controls 302. Since each of the controls 306 are also associated with selected visual controls 302, the organization of the controls 306 can be related to the structure of the controls 302.

The QA controls 302 may be used to speech-  
 20 enable both atomic controls (textbox, label, etc.) and container controls (form, panel, etc.) This provides a way of scoping behaviour and of obtaining modularity of subdialog controls. For example, the scope will allow the user of the client device to  
 25 navigate to other portions of the client side markup page without completing a dialog.

In one embodiment, "Scope" is determined as a node of the primary controls tree. The following is an example "help" command, scoped at the level of the  
 30 "Pnl1" container control, which contains two textboxes.

```

    <asp:panel id="Pnl1" ...>
      <asp:textbox id="tb1" ... />
  35    <asp:textbox id="tb2" ... />
  
```

```

5      </asp:panel>

      <Speech:QA ... >
        <Command
          id="HelpCmd1"
10       scope="Pnl1"
          type="help"
          onClientReco="GlobalGiveHelp()" >

          <Grammar src="grammars/help.gram"/>
15       </Command>
      </Speech:QA>

      <script>
        function GlobalGiveHelp() {
20         ...
        }
      </script>

```

As specified, the "help" grammar will be  
 25 active in every QA control relating to "Pnl1" and its  
 contents. The GlobalGiveHelp subroutine will execute  
 every time "help" is recognized. To override this and  
 achieve context-sensitive behavior, the same typed  
 command can be scoped to the required level of  
 30 context:

```

      <Speech:QA ... >
        <Command
          id="HelpCmd2"
35       scope="Tb2"
          type="help"
          onClientReco="SpecialGiveHelp()" >

          <Grammar src="grammars/help.gram"/>
40       </Command>
      </Speech:QA>

      <script>
        function SpecialGiveHelp() {
45         ...

```

```

5      }
      </script>

```

### Confirmation Control

The QA control 320 can also include a method for simplifying the authoring of common confirmation subdialogs. The following QA control exemplifies a typical subdialog which asks and then confirms a value:

```

15      <Speech:QA
          id="qaDepCity"
          controlsToSpeechEnable="txtDepCity"
          runat="server" >

          <!-- asking for a value -->
20      <Question      id="AskDepCity"
                  type="ask"
                  Answers="AnsDepCity" >
          <prompt> Which city? </prompt>
        </Question>
25      <Answer      id="AnsDepCity"
                  confirmThreshold="60" >
          <grammar src="grammars/depCity.gram"
        />
        </Answer>
30

          <!-- confirming the value -->
          <Confirm id="ConfirmDepCity"
                  Answers="AnsConfDepCity" >
          <prompt>
35      Did you say <value
          targetElement="txtDepCity/Text">?
          </prompt>
        </Confirm>
          <Answer id="AnsConfDepCity" >
40      <grammar
          src="grammars/YesNoDepCity.gram" />
        </Answer>

```

5           </Speech:QA>

          In this example, a user response to 'which city?' which matches the AnsDepCity grammar but whose confidence level does not exceed the confirmThreshold value will trigger the confirm control 308. More flexible methods of confirmation available to the author include mechanisms using multiple question controls and multiple answer controls.

          In a further embodiment, additional input controls related to the confirmation control include an accept control, a deny control and a correct control. Each of these controls could be activated (in a manner similar to the other controls) by the corresponding confirmation control and include grammars to accept, deny or correct results, respectively. For instance, users are likely to deny by saying "no", to accept by saying "yes" or "yes + current value" (e.g., "Do you want to go to Seattle?" "Yes, to Seattle"), to correct by saying "no" + new value (e.g., "Do you want to go to Seattle" "No, Pittsburgh").

#### Statement Control

          The statement control allows the application developer to provide an output upon execution of the client side markup when a response is not required from the user of the client device 30. An example could be a "Welcome" prompt played at

5 the beginning of execution of a client side markup page.

An attribute can be provided in the statement control to distinguish different types of information to be provided to the user of the client  
10 device. For instance, attributes can be provided to denote a warning message or a help message. These types could have different built-in properties such as different voices. If desired, different forms of statement controls can be provided, i.e. a help  
15 control, warning control, etc. Whether provided as separate controls or attributes of the statement control, the different types of statements have different roles in the dialog created, but share the fundamental role of providing information to the user  
20 of the client device without expecting an answer back.

#### Eventing

Event handlers as indicated in FIG. 11 are provided in the QA control 320, the output controls  
25 308 and the input controls 310 for actions/inactions of the user of the client device 30 and for operation of the recognition server 204 to name a few, other events are specified in Appendix B. For instance, mumbling, where the speech recognizer detects that  
30 the user has spoken but is unable to recognize the words and silence, where speech is not detected at all, are specified in the QA control 320. These events reference client-side script functions defined

5 by the author. In a multimodal application specified earlier, a simple mumble handler that puts an error message in the text box could be written as follows:

```

10      <Speech:QA
           controlsToSpeechEnable="txtDepCit
           y"      onClientNoReco="OnMumble()"
           runat="server">
           <Answer id="AnsDepCity"
           StartEvent="onMouseDown"
15           StopEvent="onMouseUp"
           >
           <grammar src="/grammars/depCities.gram"/>
           <bind value="//sml/DepCity"
           targetElement="txtCity" />
20           </Answer>
           </Speech:QA>

           <script>
           function OnMumble() {
25           txtDepCity.value="...recognition
error...";
           }
           </script>

```

### 30 Control Execution Algorithm

In one embodiment, a client-side script or module (herein referred to as "RunSpeech") is provided to the client device. The purpose of this script is to execute dialog flow via logic, which is specified in the script when executed on the client device 30, i.e. when the markup pertaining to the controls is activated for execution on the client due to values contained therein. The script allows multiple dialog turns between page requests, and therefore, is particularly helpful for control of voice-only dialogs such as through telephony browser



5 216. The client-side script RunSpeech is executed in  
a loop manner on the client device 30 until a  
completed form is submitted, or a new page is  
otherwise requested from the client device 30.

It should be noted that in one embodiment,  
10 the controls can activate each other (e.g. question  
control activating a selected answer control) due to  
values when executed on the client. However, in a  
further embodiment, the controls can "activate" each  
other in order to generate appropriate markup, in  
15 which case server-side processing may be implemented.

Generally, in one embodiment, the algorithm  
generates a dialog turn by outputting speech and  
recognizing user input. The overall logic of the  
algorithm is as follows for a voice-only scenario:

- 20 1. Find next active output companion control;
2. If it is a statement, play the statement and go  
back to 1; If it is a question or a confirm go  
to 3;
3. Collect expected answers;
- 25 4. Collect commands;
5. Play output control and listen in for input;
6. Activate recognized Answer or Command object or,  
issue an event if none is recognized;
7. Go back to 1.

30 In the multimodal case, the logic is simplified to  
the following algorithm:

1. Wait for triggering event - i.e., user tapping  
on a control;

- 5        2. Collect expected answers;
3. Listen in for input;
4. Activate recognized Answer object or, if none,  
             throw event;
5. Go back to 1.

10                The algorithm is relatively simple because,  
as noted above, controls contain built-in information  
about when they can be activated. The algorithm also  
makes use of the role of the controls in the  
dialogue. For example statements are played  
15 immediately, while questions and confirmations are  
only played once the expected answers have been  
collected.

              In a further embodiment, implicit  
confirmation can be provided whereby the system  
20 confirms a piece of information and asks a question  
at the same time. For example the system could  
confirm the arrival city of a flight and ask for the  
travel date in one utterance: "When do you want to go  
to Seattle?" (i.e. asking 'when' and implicitly  
25 confirming 'destination: Seattle'). If the user gives  
a date then the city is considered implicitly  
accepted since, if the city was wrong, users would  
have immediately challenged it. In this scenario, it  
becomes clear that the knowledge of what a user is  
30 trying to achieve is vitally important: are they  
answering the question, or are they correcting the  
value, or are they asking for help? By using the role  
of the user input in the dialogue the system can know  
when to implicitly accept a value.

5           In summary, a dialog is created due to the  
 role of the control in the dialog and the  
 relationship with other controls, wherein the  
 algorithm executes the controls and thus manages the  
 dialog. Each control contains information based on  
 10 its type which is used by the execution algorithm to  
 select (i.e. make active for execution) a given  
 control according to whether or not it serves a  
 useful purpose at that point in the dialog on the  
 client. For example, confirmation controls are only  
 15 active when there is a value to confirm and the  
 system does not have sufficient confidence in that  
 value to proceed. In a further implementation, most  
 of these built-in pieces of information can be  
 overridden or otherwise adapted by application  
 20 developers.

          The following table summarizes the  
 controls, their corresponding role in the dialog and  
 the relationship with other controls.

25

Control	Role in dialogue	Relationship with other controls
Statement	output: present information to users	(none)
Question	output: ask question	selects expected Answers as a response
Confirmation	output: confirm a value obtained from the user	selects potential input controls as a response, typically Accept, Deny, Correct

Control	Role in dialogue	Relationship with other controls
Answer	input: provide an answer to a question	selected by Question/Confirmation
Command	input: seek to repair a dialog, or change the topic of conversation	scoped to other controls
Accept	input: confirm a value in response to a confirmation	selected by a confirmation
Deny	input: deny a value in response to a confirmation	selected by a confirmation
Correct	input: correct a value in response to a confirmation	selected by a confirmation
QA		(wrapper: contains all the controls related to a specific topic)

5

The use of these controls may be explained with an illustration of a simple human/computer dialog. In the dialog below, each dialog turn on the part of the System or the User is characterized according to the control (indicated in parentheses) which reflects its purpose in the dialog.

1. System (Statement): "Welcome to the travel booking service".
2. System (Question): "Where would you like to go?"
- 15 3. User (Answer): "San Francisco."
4. System (Confirmation): "Did you say Seattle?"
5. User (Deny): "No."
6. System (Question): "Where would you like to go?"
7. User (Answer): "San Francisco."

- 5 8. System (Confirmation): "Did you say Seattle?"  
 9. User (Correct): "I said San Francisco."  
 10. System (Confirmation): "Did you say San Francisco?"  
 11. User (Correct): "Yes."  
 10 12. System (Question): "When would you like to leave?"  
 13. User (Command): "Help."

Turn 1 is a statement on the part of the System. Since a statement control activates no answer controls in response, the system does not expect  
 15 input. The system goes on to activate a question control at turn 2. This in turn activates a set of possible answer controls, including one which holds a grammar containing the cities available through the service, including "San Francisco", "Seattle", etc.,  
 20 which permits the user to provide such a city in turn 3.

The user's turn 3 is misrecognized by the system. Although the system believes it has a value  
 25 from an answer control for the city, its confidence in that value is low (rightly so, since it has recognized incorrectly). This low confidence value in a just-received answer control is sufficient information for RunSpeech to trigger a confirmation  
 30 control on the part of the system, as generated at turn 4. The confirmation control in turn activates a deny control, a correct control and an accept control and makes their respective grammars available to recognize the user's next turn. User turns 5, 9 and

5 11 illustrate example responses for these controls.  
 Turn 5 of the user simply denies the value "no". This  
 has the effect of removing the value from the system,  
 so the next action of RunSpeech is to ask the  
 question again to re-obtain the value (turn 6).

10 Turns 7 and 8 return us to a confirmation  
 control as with 3 and 4.

User turn 9 is a correct control, which has  
 again been activated as a possible response to the  
 confirmation control. A correct control not only  
 15 denies the value undergoing confirmation, it also  
 provides a new value. So user turn 9 is recognized by  
 the system as a correct control with a new value  
 which, correctly this time, is recognized as "San  
 Francisco".

20 The system's confidence in the new value is  
 low, however, and yet another confirmation control is  
 generated at turn 10. This in turn activates accept,  
 deny and correct controls in response, and user turn  
 11 ("Yes" matches an accept control grammar. The  
 25 recognition of the accept control has the effect of  
 'grounding' the system's belief in the value which it  
 is trying to obtain, and so RunSpeech is now able to  
 select other empty values to obtain. In turn 12, a  
 new question control is output which asks for a date  
 30 value. The user's response this time (turn 13) is a  
 command: "help". Command controls are typically  
 activated in global fashion, that is, independently  
 of the different question controls and confirmation  
 controls on the part of the system. In this way the

5 user is able to ask for help at any time, as he does  
in turn 13. Command controls may also be more  
sensitively enabled by a mechanism that scopes their  
activation according to which part of the primary  
control structure is being talked about.

10 Referring back to the algorithm, in one  
exemplary embodiment, the client-side script  
RunSpeech examines the values inside each of the  
primary controls and an attribute of the QA control,  
and any selection test of the QA controls on the  
15 current page, and selects a single QA control for  
execution. For example, within the selected QA  
control, a single question and its corresponding  
prompt are selected for output, and then a grammar is  
activated related to typical answers to the  
20 corresponding question. Additional grammars may also  
be activated, in parallel, allowing other commands  
(or other answers), which are indicated as being  
allowable. Assuming recognition has been made and any  
further processing on the input data is complete, the  
25 client-side script RunSpeech will begin again to  
ascertain which QA control should be executed next.  
An exemplary implementation and algorithm of  
RunSpeech is provided in Appendix B.

It should be noted that the use of the controls  
30 and the RunSpeech algorithm or module is not limited  
to the client/server application described above, but  
rather can be adapted for use with other application  
abstractions. For instance, an application such as  
VoiceXML, which runs only on the client device 30,

5 could conceivably include further elements or  
controls such as question and answer provided above  
as part of the VoiceXML browser and operating in the  
same manner. In this case the mechanisms of the  
RunSpeech algorithm described above could be executed  
10 by default by the browser without the necessity for  
extra script. Similarly, other platforms such as  
finite state machines can be adapted to include the  
controls and RunSpeech algorithm or module herein  
described.

15

#### Synchronization

As noted above, the companion controls 306  
are associated with the primary controls 302 (the  
existing controls on the page). As such the companion  
20 controls 306 can re-use the business logic and  
presentation capabilities of the primary controls  
302. This is done in two ways: storing values in the  
primary controls 302 and notifying the primary  
controls of the changes 302.

25

The companion controls 306 synchronize or  
associate their values with the primary controls 302  
via the mechanism called binding. Binding puts values  
retrieved from recognizer into the primary controls  
302, for example putting text into a textbox, herein  
30 exemplified with the answer control. Since primary  
controls 302 are responsible for visual presentation,  
this provides visual feedback to the users in  
multimodal scenarios.



5           The companion controls 306 also offer a  
mechanism to notify the primary controls 302 that  
they have received an input via the recognizer. This  
allows the primary controls 302 to take actions, such  
as invoking the business logic. (Since the  
10 notification amounts to a commitment of the companion  
controls 306 to the values which they write into the  
primary controls 302, the implementation provides a  
mechanism to control this notification with a fine  
degree of control. This control is provided by the  
15 *RejectThreshold* and *ConfirmThreshold* properties on  
the *answer* control, which specify numerical acoustic  
confidence values below which the system should  
respectively reject or attempt to confirm a value.)

          Although the present invention has been  
20 described with reference to preferred embodiments,  
workers skilled in the art will recognize that changes  
may be made in form and detail without departing from  
the spirit and scope of the invention.

## APPENDIX A

### 1 Introduction

---

The following tags are a set of markup elements that allows a document to use speech as an input or output medium. The tags are designed to be self-contained XML that can be imbedded into any SGML derived markup languages such as HTML, XHTML, cHTML, SMIL, WML and the like. The tags used herein are similar to SAPI 5.0, which are known methods available from Microsoft Corporation of Redmond, Washington. The tags, elements, events, attributes, properties, return values, etc. are merely exemplary and should not be considered limiting. Although exemplified herein for speech and DTMF recognition, similar tags can be provided for other forms of recognition.

The main elements herein discussed are:

20	<code>&lt;prompt ...&gt;</code>	for speech synthesis configuration and prompt playing
	<code>&lt;reco ...&gt;</code>	for recognizer configuration and recognition execution and post-processing
	<code>&lt;grammar ...&gt;</code>	for specifying input grammar resources
25	<code>&lt;bind ...&gt;</code>	for processing of recognition results
	<code>&lt;dtmf ...&gt;</code>	for configuration and control of DTMF

### 2 Reco

---

The Reco element is used to specify possible user inputs and a means for dealing with the input results.

As such, its main elements are <grammar> and <bind>, and it contains resources for configuring recognizer properties.

- 5 Reco elements are activated programmatically in uplevel browsers via Start and Stop methods, or in SMIL-enabled browsers by using SMIL commands. They are considered active declaratively in downlevel browsers (i.e. non script-supporting browsers) by their  
10 presence on the page. In order to permit the activation of multiple grammars in parallel, multiple Reco elements may be considered active simultaneously.

- Reco may also take a particular mode - 'automatic',  
15 'single' or 'multiple' - to distinguish the kind of recognition scenarios which they enable and the behaviour of the recognition platform.

### **2.1 Reco content**

- The Reco element contains one or more grammars and  
20 optionally a set of bind elements which inspect the results of recognition and copy the relevant portions to values in the containing page.

- In uplevel browsers, Reco supports the programmatic  
25 activation and deactivation of individual grammar rules. Note also that all top-level rules in a grammar are active by default for a recognition context.

### 2.1.1 <grammar> element

The grammar element is used to specify grammars, either inline or referenced using the src attribute. At least one grammar (either inline or referenced) is typically specified. Inline grammars can be text-based grammar formats, while referenced grammars can be text-based or binary type. Multiple grammar elements may be specified. If more than one grammar element is specified, the rules within grammars are added as extra rules within the same grammar. Any rules with the same name will be overwritten.

#### Attributes:

- **src:** Optional if inline grammar is specified. URI of the grammar to be included. Note that all top-level rules in a grammar are active by default for a recognition context.
- **langID:** Optional. String indicating which language speech engine should use. The string format follows the xml:lang definition. For example, langID="en-us" denotes US English. This attribute is only effective when the langID is not specified in the grammar URI. If unspecified, defaults to US English.

If the langID is specified in multiple places then langID follows a precedence order from the lowest scope - remote grammar file (i.e language

id is specified within the grammar file) followed by grammar element followed by reco element.

```
5      <grammar src="FromCity.xml" />
      or
      <grammar>
        <rule toplevel="active">
          <p>from </p>
          <ruleref name="cities" />
10      </rule>
        <rule name="cities">
          <l>
            <p> Cambridge </p>
            <p> Seattle </p>
15          <p> London </p>
          </l>
        </rule>
      </grammar>
```

- 20 If both a src-referenced grammar and an inline grammar are specified, the inline rules are added to the referenced rules, and any rules with the same name will be overwritten.

### 2.1.2 <bind> element

- 25 The bind element is used to bind values from the recognition results into the page.

The recognition results consumed by the bind element can be an XML document containing a semantic markup language (SML) for specifying recognition results. Its contents include semantic values, actual words spoken, and confidence scores. SML could also include alternate recognition choices (as in an N-best recognition result). A sample SML document for the

utterance "I'd like to travel from Seattle to Boston"  
is illustrated below:

```
5      <sml confidence="40">
      <travel text="I'd like to travel from
      Seattle to Boston">
      <origin_city confidence="45"> Seattle
</origin_city>
      <dest_city confidence="35"> Boston
10 </dest_city>
      </travel>
      </sml>
```

Since an in-grammar recognition is assumed to produce  
15 an XML document - in semantic markup language, or SML  
- the values to be bound from the SML document are  
referenced using an XPath query. And since the  
elements in the page into which the values will be  
bound should be uniquely identified (they are  
20 likely to be form controls), these target elements are  
referenced directly.

#### Attributes:

- **targetElement:** Required. The element to which the  
25 value content from the SML will be assigned (as  
in W3C SMIL 2.0).
- **targetAttribute:** Optional. The attribute of the  
target element to which the value content from  
the SML will be assigned (as with the  
30 **attributeName** attribute in SMIL 2.0). If  
unspecified, defaults to "value".

- **test:** Optional. An XML Pattern (as in the W3C XML DOM specification) string indicating the condition under which the recognition result will be assigned. Default condition is true.
- 5 • **value:** Required. An XPATH (as in the W3C XML DOM specification) string that specifies the value from the recognition result document to be assigned to the target element.

10 *Example:*

So given the above SML return, the following reco element uses bind to transfer the values in origin\_city and dest\_city into the target page elements txtBoxOrigin and txtBoxDest:

15

```
<input name="txtBoxOrigin" type="text"/>
<input name="txtBoxDest" type="text" />
```

```
<reco id="travel">
```

20

```
<grammar src="./city.xml" />
```

```
<bind      targetElement="txtBoxOrigin"
          value="//origin_city" />
```

```
<bind      targetElement="txtBoxDest"
          value="//dest_city" />
```

25

```
</reco>
```

This binding may be conditional, as in the following example, where a test is made on the confidence

attribute of the dest\_city result as a pre-condition to the bind operation:

```
5      <bind targetElement="txtBoxDest"
        value="//dest_city"
        test="/sml/dest_city[@confidence $gt$ 40]"
      />
```

10 The bind element is a simple declarative means of processing recognition results on downlevel or uplevel browsers. For more complex processing, the reco DOM object supported by uplevel browsers implements the onReco event handler to permit programmatic script analysis and post-processing of the recognition  
15 return.

## **2.2 Attributes and properties**

The following attributes are supported by all browsers, and the properties by uplevel browsers.

### **2.2.1 Attributes**

20 The following attributes of Reco are used to configure the speech recognizer for a dialog turn.

- **initialTimeout:** Optional. The time in  
25 milliseconds between start of recognition and the detection of speech. This value is passed to the recognition platform, and if exceeded, an onSilence event will be provided from the recognition platform (see 2.4.2). If not



specified, the speech platform will use a default value.

- 5     • **babbleTimeout**: Optional. The period of time in milliseconds in which the recognizer must return a result after detection of speech. For  
10     recos in automatic and single mode, this applies to the period between speech detection and the stop call. For recos in 'multiple' mode, this timeout applies to the period  
15     between speech detection and each recognition return - i.e. the period is restarted after each return of results or other event. If exceeded, different events are thrown according to whether an error has occurred or  
20     not. If the recognizer is still processing audio - eg in the case of an exceptionally long utterance - the onNoReco event is thrown, with status code 13 (see 2.4.4). If the timeout is exceeded for any other reason, however, a recognizer error is more likely, and the onTimeout event is thrown. If not specified, the speech platform will default to an internal value.
- 25     • **maxTimeout**: Optional. The period of time in milliseconds between recognition start and results returned to the browser. If exceeded, the onTimeout event is thrown by the browser - this caters for network or recognizer failure in distributed environments. For recos in

'multiple' mode, as with babbleTimeout, the period is restarted after the return of each recognition or other event. Note that the maxTimeout attribute should be greater than or equal to the sum of initialTimeout and babbleTimeout. If not specified, the value will be a browser default.

- **endSilence:** Optional. For Recos in automatic mode, the period of silence in milliseconds after the end of an utterance which must be free of speech after which the recognition results are returned. Ignored for recos of modes other than automatic. If unspecified, defaults to platform internal value.
- **reject:** Optional. The recognition rejection threshold, below which the platform will throw the 'no reco' event. If not specified, the speech platform will use a default value. Confidence scores range between 0 and 100 (integer). Reject values lie in between.
- **server:** Optional. URI of speech platform (for use when the tag interpreter and recognition platform are not co-located). An example value might be *server=protocol://yourspeechplatform*. An application writer is also able to provide speech platform specific settings by adding a querystring to the URI string, eg *protocol://yourspeechplatform?bargainEnergyThreshold=0.5*.

- **langID:** Optional. String indicating which language speech engine should use. The string format follows the xml:lang definition. For example, langID="en-us" denotes US English. This attribute is only effective when the langID is not specified in the grammar element (see 2.1.1).
- **mode:** Optional. String specifying the recognition mode to be followed. If unspecified, defaults to "automatic" mode.

### 2.2.2 Properties

The following properties contain the results returned by the recognition process (these are supported by uplevel browsers).

- **recoResult** Read-only. The results of recognition, held in an XML DOM node object containing semantic markup language (SML), as described in 2.1.2, In case of no recognition, the property returns null.
- **text** Read-only. A string holding the text of the words recognized (i.e., a shorthand for contents of the text attribute of the highest level element in the SML recognition return in recoResult.
- **status:** Read-only. Status code returned by the recognition platform. Possible values are 0 for successful recognition, or the failure values -1

to -4 (as defined in the exceptions possible on  
the Start method (section 2.3.1) and Activate  
method (section 2.3.4)), and statuses -11 to -15  
set on the reception of recognizer events (see  
5 2.4).

### 2.3 Object methods

Reco activation and grammar activation may be  
controlled using the following methods in the Reco's  
DOM object. With these methods, uplevel browsers can  
10 start and stop Reco objects, cancel recognitions in  
progress, and activate and deactivate individual  
grammar top-level rules (uplevel browsers only).

#### 2.3.1 Start

The Start method starts the recognition process, using  
15 as active grammars all the top-level rules for the  
recognition context which have not been explicitly  
deactivated.

#### Syntax:

20 Object.Start( )

#### Return value:

None.

#### Exception:

25 The method sets a non-zero status code and  
fires an onNoReco event when fails. Possible  
failures include no grammar (reco status = -  
1), failure to load a grammar, which could  
be a variety of reasons like failure to

compile grammar, non-existent URI (reco  
status = -2), or speech platform errors  
(reco status = -3).

### 2.3.2 Stop

- 5 The Stop method is a call to end the recognition  
process. The Reco object stops recording audio, and  
the recognizer returns recognition results on the  
audio received up to the point where recording was  
stopped. All the recognition resources used by Reco  
10 are released, and its grammars deactivated. (Note that  
this method need not be used explicitly for typical  
recognitions in automatic mode, since the recognizer  
itself will stop the reco object on endpoint detection  
after recognizing a complete sentence.) If the Reco  
15 has not been started, the call has no effect.

#### Syntax:

Object.Stop( )

#### Return value:

- 20 None.

#### Exception:

None.

### 2.3.3 Cancel

- 25 The Cancel method stops the audio feed to the  
recognizer, deactivates the grammar and releases the  
recognizer and discards any recognition results. The  
browser will disregard a recognition result for

canceled recognition. If the recognizer has not been started, the call has no effect.

**Syntax:**

5           Object.Cancel( )

**Return value:**

None.

**Exception:**

None.

10

**2.3.4           Activate**

The Activate method activates a top-level rule in the context free grammar (CFG). Activation must be called before recognition begins, since it will have no effect during a 'Started' recognition process. Note that all the grammar top-level rules for the recognition context which have not been explicitly deactivated are already treated as active.

20

**Syntax:**

Object.Activate(strName);

**Parameters:**

- o **strName:** Required. Rule name to be activated.

25

**Return value:**

None.

**Exception:**

None.

### 2.3.5 Deactivate

The method deactivates a top-level rule in the grammar. If the rule does not exist, the method has no effect.

5

#### Syntax:

Object.Deactivate(strName);

#### Parameters:

- o **strName:** Required. Rule name to be  
10 deactivated. An empty string deactivates all rules.

#### Return value

None.

#### Exception

15 None.

### 2.4 Reco events

The Reco DOM object supports the following events, whose handlers may be specified as attributes of the reco element.

#### 20 2.4.1 onReco:

This event gets fired when the recognizer has a recognition result available for the browser. For  
reco in automatic mode, this event stops the  
recognition process automatically and clears  
25 resources (see 2.3.2). OnReco is typically used  
for programmatic analysis of the recognition

result and processing of the result into the page.

**Syntax:**

5

Inline HTML	<code>&lt;Reco onReco ="handler" &gt;</code>
Event property	<code>Object.onReco = handler;</code> <code>Object.onReco =</code> <code>GetRef("handler");</code>

**Event Object Info:**

Bubbles	No
To invoke	User says something
Default action	Return recognition result object

**Event Properties:**

10

Although the event handler does not receive properties directly, the handler can query the event object for data (see the use of the event object in the example below).

15

**Example**

The following XHTML fragment uses onReco to call a script to parse the recognition outcome and assign the values to the proper fields.

20

```
<input name="txtBoxOrigin" type="text" />
<input name="txtBoxDest" type="text" />
```



```
<reco onReco="processCityRecognition()" />
    <grammar src="/grammars/cities.xml" />
</reco>
```

```
5    <script><![CDATA[
        function processCityRecognition () {
            smlResult =
event.srcElement.recoResult;

10            origNode =
smlResult.selectSingleNode("//origin_city");
            if (origNode != null)
txtBoxOrigin.value = origNode.text;

15            destNode =
smlResult.selectSingleNode("//dest_city");
            if (destNode != null) txtBoxDest.value
= destNode.text;
        }
20    ]]></script>
```

#### 2.4.2 onSilence:

onSilence handles the event of no speech detected by the recognition platform before the duration of time specified in the initialTimeout attribute on the Reco (see 2.2.1). This event cancels the recognition process automatically for the automatic recognition mode.

##### Syntax:

Inline HTML	<reco onSilence="handler" ...>
-------------	--------------------------------

Event property (in ECMAScript)	Object.onSilence = <i>handler</i> Object.onSilence = GetRef("handler");
--------------------------------	---

**Event Object Info:**

Bubbles	No
To invoke	Recognizer did not detect speech within the period specified in the initialTimeout attribute.
Default action	Set status = -11

**Event Properties:**

- 5        Although the event handler does not receive properties directly, the handler can query the event object for data.

**2.4.3            onTimeout**

10       onTimeout handles two types of event which typically reflect errors from the speech platform.

- 15       • It handles the event thrown by the tags interpreter which signals that the period specified in the maxtime attribute (see 2.2.1) expired before recognition was completed. This event will typically reflect problems that could occur in a distributed architecture.
- It also handles (ii) the event thrown by the speech recognition platform when recognition has

begun but processing has stopped without a recognition within the period specified by babbleTimeout (see 2.2.1).

- 5 This event cancels the recognition process automatically.

**Syntax:**

Inline HTML	<reco onTimeout="handler" ...>
Event property (in ECMAScript)	Object.onTimeOut = handler Object.onTimeOut = GetRef("handler");

- 10 **Event Object Info:**

Bubbles	No
To invoke	Thrown by the browser when the period set by the maxtime attribute expires before recognition is stopped.
Default action	Set reco status to -12.

**Event Properties:**

- 15 Although the event handler does not receive properties directly, the handler can query the event object for data.

**2.4.4 onNoReco:**

onNoReco is a handler for the event thrown by the speech recognition platform when it is unable to

return valid recognition results. The different cases in which this may happen are distinguished by status code. The event stops the recognition process automatically.

5

**Syntax:**

Inline HTML	<code>&lt;Reco onNoReco ="handler" &gt;</code>
Event property	<code>Object.onNoReco = handler;</code> <code>Object.onNoReco =</code> <code>GetRef("handler");</code>

**Event Object Info:**

Bubbles	No
To invoke	Recognizer detects sound but is unable to interpret the utterance.

Default action	<p>Set status property and return null recognition result. Status codes are set as follows:</p> <p><b>status -13:</b> sound was detected but no speech was able to be interpreted;</p> <p><b>status -14:</b> some speech was detected and interpreted but rejected with insufficient confidence (for threshold setting, see the reject attribute in 2.2.1).</p> <p><b>status -15:</b> speech was detected and interpreted, but a complete recognition was unable to be returned between the detection of speech and the duration specified in the babbleTimeout attribute (see 2.2.1).</p>
-------------------	--

#### Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

5

### 3 Prompt

---

The prompt element is used to specify system output. Its content may be one or more of the following:

10

- inline or referenced text, which may be marked up with prosodic or other speech output information;
  - variable values retrieved at render time from the containing document;
- 5      • links to audio files.

Prompt elements may be interpreted declaratively by downlevel browsers (or activated by SMIL commands), or by object methods on uplevel browsers.

10    **3.1 Prompt content**

The prompt element contains the resources for system output, either as text or references to audio files, or both.

- 15    Simple prompts need specify only the text required for output, eg:

```
<prompt id="Welcome">
```

```
    Thank you for calling ACME weather report.
```

20    </prompt>

This simple text may also contain further markup of any of the kinds described below.

**3.1.1            Speech Synthesis markup**

- 25    Any format of speech synthesis markup language can be used inside the prompt element. (This format may be specified in the 'tts' attribute described in 3.2.1.)

The following example shows text with an instruction to emphasize certain words within it:

```

    <prompt id="giveBalance">
5      You have <emph> five dollars </emph> left in
      your account.
    </prompt>
```

### 3.1.2 Dynamic content

The actual content of the prompt may need to be  
10 computed on the client just before the prompt is  
output. In order to confirm a particular value, for  
example, the value needs to be dereferenced in a  
variable. The value element may be used for this  
purpose.

15

#### **Value Element**

**value:** Optional. Retrieves the values of an element in  
the document.

20 **Attributes:**

- **targetElement:** Optional. Either href or  
targetElement must be specified. The id of the  
element containing the value to be retrieved.
- **targetAttribute:** Optional. The attribute of the  
25 element from which the value will be retrieved.
- **href:** Optional. The URI of an audio segment. href  
will override targetElement if both are present.

The targetElement attribute is used to reference an element within the containing document. The content of the element whose id is specified by targetElement is inserted into the text to be synthesized. If the  
5 desired content is held in an attribute of the element, the targetAttribute attribute may be used to specify the necessary attribute on the targetElement. This is useful for dereferencing the values in HTML form controls, for example. In the following  
10 illustration, the "value" attributes of the "txtBoxOrigin" and "txtBoxDest" elements are inserted into the text before the prompt is output

```

    <prompt id="Confirm">
15      Do you want to travel from
        <value targetElement="txtBoxOrigin"
targetAttribute="value" />
        to
        <value targetElement="txtBoxDest"
20 targetAttribute="value" />
        ?
    </prompt>
```

### 3.1.3 Audio files

The value element may also be used to refer to a pre-  
25 recorded audio file for playing instead of, or within, a synthesized prompt. The following example plays a beep at the end of the prompt:

```
<prompt>
```

After the beep, please record your message.



```
        <value href="/wav/beep.wav" />
    </prompt>
```

#### 3.1.4 Referenced prompts

Instead of specifying content inline, the src  
5 attribute may be used with an empty element to  
reference external content via URI, as in:

```
        <prompt id="Welcome"
10    src="/ACMEWeatherPrompts#Welcome" />
```

The target of the src attribute can hold any or all of  
the above content specified for inline prompts.

#### 3.2 Attributes and properties

The prompt element holds the following attributes  
15 (downlevel browsers) and properties (downlevel and  
uplevel browsers).

##### 3.2.1 Attributes

- **tts:** Optional. The markup language type for  
text-to-speech synthesis. Default is "SAPI 5".
- 20 • **src:** Optional if an inline prompt is  
specified. The URI of a referenced prompt (see  
3.1.4).
- 25 • **bargein:** Optional. Integer. The period of time  
in milliseconds from start of prompt to when  
playback can be interrupted by the human  
listener. Default is infinite, i.e., no  
bargein is allowed. Bargein=0 allows immediate

bargein. This applies to whichever kind of barge-in is supported by platform. Either keyword or energy-based bargein times can be configured in this way, depending on which is enabled at the time the reco is started.

- **prefetch:** Optional. A Boolean flag indicating whether the prompt should be immediately synthesized and cached at browser when the page is loaded. Default is false.

### 10 3.2.2 Properties

Uplevel browsers support the following properties in the prompt's DOM object.

- **bookmark:** Read-only. A string object recording the text of the last synthesis bookmark encountered.
- **status:** Read-only. Status code returned by the speech platform.

### 3.3 Prompt methods

20 Prompt playing may be controlled using the following methods in the prompt's DOM object. In this way, uplevel browsers can start and stop prompt objects, pause and resume prompts in progress, and change the speed and volume of the synthesized speech.

#### 25 3.3.1 Start

Start playback of the prompt. Unless an argument is given, the method plays the contents of the

object. Only a single prompt object is considered 'started' at a given time, so if Start is called in succession, all playbacks are played in sequence.

5

**Syntax:**

```
Object.Start([strText] );
```

**Parameters:**

- o **strText:** the text to be sent to the synthesizer. If present, this argument overrides the contents of the object.

10

**Return value:**

None.

**Exception:**

- Set status = -1 and fires an onComplete event if the audio buffer is already released by the server.

15

### 3.3.2 Pause

- Pause playback without flushing the audio buffer.
- This method has no effect if playback is paused or stopped.

20

**Syntax:**

```
Object.Pause( );
```

25

**Return value:**

None.

**Exception:**

None.

### 3.3.3 Resume

Resume playback without flushing the audio buffer. This method has no effect if playback has not been paused.

5

**Syntax:**

Object.Resume( );

**Return value:**

None.

10

**Exception:**

Throws an exception when resume fails.

### 3.3.4 Stop

Stop playback, if not already, and flush the audio buffer. If the playback has already been stopped, the method simply flushes the audio buffer.

15

**Syntax:**

Object.Stop( );

**Return value:**

20

None.

**Exception:**

None.

### 3.3.5 Change

Change speed and/or volume of playback. Change may be called during playback.

25

**Syntax:**

Object.Change(speed, volume);

**Parameters:**

- o **speed:** Required. The factor to change.  
5      Speed=2.0 means double the current rate,  
         speed=0.5 means halve the current rate,  
         speed=0 means to restore the default value.
- o **volume:** Required. The factor to change.  
10      Volume=2.0 means double the current volume,  
         volume =0.5 means halve the current volume,  
         volume =0 means to restore the default  
         value.

**Return value:**

None.

15      **Exception:**

None.

**3.3.6      Prompt control example**

The following example shows how control of the prompt  
using the methods above might be authored for a  
20 platform which does not support a keyword barge-in  
mechanism.

```
<html>
<title>Prompt control</title>
25<head>
  <script>
    <!--
      function checkKWBargein() {
        news.change(1.0, 0.5); // turn down the
30 volume while verifying
        if (keyword.text == "") { // result is below
threshold
```

```

                                news.change(1.0, 2.0); // restore the
volume
                                keyword.Start(); // restart the
recognition
5      } else {
                                news.Stop(); // keyword detected! Stop
the prompt
                                // Do whatever that is necessary
                                }
10     }
    //
    </script>
    <script for="window" event="onload">
        <!--
15         news.Start(); keyword.Start();
        //
        </script>
        </head>
    <body>
20 <prompt id="news" bargein="0">
    Stocks turned in another lackluster performance
    Wednesday as investors received little incentive to
    make any big moves ahead of next week's Federal
    Reserve meeting. The tech-heavy Nasdaq Composite Index
25 dropped 42.51 points to close at 2156.26. The Dow
    Jones Industrial Average fell 17.05 points to 10866.46
    after an early-afternoon rally failed.
    - <!--
    </prompt>
30 <reco      id="keyword"
        reject="70"
        onReco="checkKWBargein()" >
        <grammar
        src=http://denali/news bargein grammar.xml />
35 </reco>
    </body>
    </html>
```

### 3.4 Prompt events

The prompt DOM object supports the following events, whose handlers may be specified as attributes of the prompt element.

#### 5 3.4.1 onBookmark

Fires when a synthesis bookmark is encountered.  
The event does not pause the playback.

##### Syntax:

Inline HTML	<code>&lt;prompt onBookmark="handler" ...&gt;</code>
Event property	<code>Object.onBookmark = handler</code> <code>Object.onBookmark =</code> <code>GetRef("handler");</code>

10

##### Event Object Info:

Bubbles	No
To invoke	A bookmark in the rendered string is encountered
Default action	Returns the bookmark string

##### Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

15

### 3.4.2 onBargein:

5 Fires when a user's barge-in event is detected.  
(Note that determining what constitutes a barge-in event, eg energy detection or keyword recognition, is up to the platform.) A specification of this event handler does not automatically turn the barge-in on.

#### Syntax:

Inline HTML	<code>&lt;prompt onBargein="handler" ...&gt;</code>
Event property	<code>Object.onBargein = handler</code> <code>Object.onBargein = GetRef("handler");</code>

#### Event Object Info:

Bubbles	No
To invoke	A bargein event is encountered
Default action	None

#### Event Properties:

15 Although the event handler does not receive properties directly, the handler can query the event object for data.

### 3.4.3 onComplete:

Fires when the prompt playback reaches the end or exceptions (as defined above) are encountered.



**Syntax:**

Inline HTML	<code>&lt;prompt onComplete="handler" ...&gt;</code>
Event property	<code>Object. onComplete = handler Object. onComplete = GetRef("handler");</code>

**Event Object Info:**

Bubbles	No
To invoke	A prompt playback completes
Default action	Set status = 0 if playback completes normally, otherwise set status as specified above.

5

**Event Properties:**

Although the event handler does not receive properties directly, the handler can query the event object for data.

10    **3.4.4            Using bookmarks and events**

15    The following example shows how bookmark events can be used to determine the semantics of a user response - either a correction to a departure city or the provision of a destination city - in terms of when bargein happened during the prompt output. The onBargein handler calls a script which sets a global 'mark' variable to the last bookmark encountered in the prompt, and the value of this 'mark' is used in

the reco's postprocessing function ('heard') to set the correct value.

```
5      <script><![CDATA[
      var mark;
      function interrupt( ) {
          mark = event.srcElement.bookmark;
      }
      function ProcessCityConfirm() {
10          confirm.stop(); // flush the audio
      buffer
          if (mark == "mark_origin_city")
              txtBoxOrigin.value =
event.srcElement.text;
15          else
              txtBoxDest.value =
event.srcElement.text;
      }
      ]]></script>
20      <body>
      <input name="txtBoxOrigin" value="Seattle"
type="text"/>
      <input name="txtBoxDest" type="text" />
      ...
25      <prompt id="confirm" onBargein="interrupt()"
bargein="0">
          From <bookmark mark="mark_origin_city" />
          <value targetElement="origin"
targetAttribute="value" />,
30      please say <bookmark mark="mark_dest_city"
/> the
          destination city you want to travel to.
      </prompt>
      <reco onReco="ProcessCityConfirm()" >
35      <grammar src="/grm/1033/cities.xml" />
      </reco>
      ...
      </body>
```

## 4 DTMF

---

Creates a DTMF recognition object. The object can be instantiated using inline markup language syntax or in scripting. When activated, DTMF can cause prompt  
5 object to fire a barge-in event. It should be noted the tags and eventing discussed below with respect to DTMF recognition and call control discussed in Section 5 generally pertain to interaction between the voice browser 216 and media server 214.

### 10 4.1 Content

- **dtmfgrammar:** for inline grammar.
- **bind:** assign DTMF conversion result to proper field.

15

#### Attributes:

- **targetElement:** Required. The element to which a partial recognition result will be assigned to (cf. same as in W3C SMIL 2.0).
- 20 • **targetAttribute:** the attribute of the target element to which the recognition result will be assigned to (cf. same as in SMIL 2.0). Default is "value".
- 25 • **test:** condition for the assignment. Default is true.

**Example 1:** map keys to text.

```

    <input type="text" name="city"/>
    <DTMF id="city_choice" timeout="2000"
numDigits="1">
    <dtmfgrammar>
5        <key value="1">Seattle</key>
        <key value="2">Boston</key>
    </dtmfgrammar>
    <bind targetElement="city"
targetAttribute="value" />
10    </DTMF>
```

When "city\_choice" is activated, "Seattle" will be assigned to the input field if the user presses 1, "Boston" if 2, nothing otherwise.

15

**Example 2:** How DTMF can be used with multiple fields.

```

    <input type="text" name="area_code" />
    <input type="text" name="phone_number" />
20    <DTMF id="areacode" numDigits="3"
onReco="extension.Activate()">
    <bind targetElement="area_code" />
    </DTMF>
    <DTMF id="extension" numDigits="7">
25    <bind targetElement="phone_number" />
    </DTMF>
```

This example demonstrates how to allow users entering into multiple fields.

30

**Example 3:** How to allow both speech and DTMF inputs and disable speech when user starts DTMF.

```

35    <input type="text" name="credit_card_number" />
    <prompt onBookmark="dtmf.Start(); speech.Start()"
        bargein="0">
        Please say <bookmark name="starting" />
        or enter your credit card number now
```

```

    </prompt>
    <DTMF id="dtmf" escape="#" length="16"
interdigitTimeout="2000"
    onkeypress="speech.Stop()">
5    <bind targetElement="credit_card_number" />
    </DTMF>
    <reco id="speech" >
        <grammar src="/grm/1033/digits.xml" />
        <bind targetElement="credit_card_number" />
10    </reco>

```

## 4.2 Attributes and properties

### 4.2.1 Attributes

- **dtmfgrammar**: Required. The URI of a DTMF grammar.

### 15 4.2.2 Properties

- **DTMFgrammar** Read-Write.

An XML DOM Node object representing DTMF to string conversion matrix (also called DTMF grammar). The default grammar is

```

20    <dtmfgrammar>
        <key value="0">0</key>
        <key value="1">1</key>
        ...
25    <key value="9">9</key>
        <key value="*">*</key>
        <key value="#">#</key>
    </dtmfgrammar >

```

### 30 • **flush**

Read-write, a Boolean flag indicating whether to automatically flush the DTMF buffer on the underlying telephony interface card before

activation. Default is false to enable type-ahead.

- **escape**

5      Read-Write. The escape key to end the DTMF reading session. Escape key is one key.

- **numDigits**

10      Read-Write. Number of key strokes to end the DTMF reading session. If both escape and length are specified, the DTMF session is ended when either condition is met.

- **dtmfResult**

15      Read-only string, storing the DTMF keys user has entered. Escape is included in result if typed.

- **text**

20      Read-only string storing white space separated token string, where each token is converted according to DTMF grammar.

- **initialTimeout**

25      Read-Write. Timeout period for receiving the first DTMF keystroke, in milliseconds. If unspecified, defaults to the telephony platform's internal setting.

- **interdigitTimeout**

Read-Write. Timeout period for adjacent DTMF  
keystrokes, in milliseconds. If unspecified,  
defaults to the telephony platform's internal  
5 setting.

#### **4.3 Object methods:**

##### **4.3.1 Start**

Enable DTMF interruption and start a DTMF reading  
session.

10

**Syntax:**

Object.Start( );

**Return value:**

None

15

**Exception:**

None

##### **4.3.2 Stop**

Disable DTMF. The key strokes entered by the  
20 user, however, remain in the buffer.

**Syntax:**

Object.Stop( );

**Return value:**

25

None

**Exception:**

None

### 4.3.3 Flush

Flush the DTMF buffer. Flush can not be called during a DTMF session..

#### 5 Syntax:

```
Object.Flush( );
```

#### Return value:

None

#### Exception:

10 None

## 4.4 Events

### 4.4.1 onkeypress

15 Fires when a DTMF key is press. This overrides the default event inherited from the HTML control. When user hits the escape key, the onRec event fires, not onKeyPress.

#### Syntax:

Inline HTML	<DTMF onkeypress="handler" ...>
Event property	Object.onkeypress = handler Object.onkeypress = GetRef("handler");

20

#### Event Object Info:

Bubbles	No
---------	----



To invoke	Press on the touch-tone telephone key pad
Default action	Returns the key being pressed

#### Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

5

#### 4.4.2 onReco

Fires when a DTMF session is ended. The event disables the current DTMF object automatically.

10

#### Syntax:

Inline HTML	<DTMF onReco="handler" ...>
Event property	Object.onReco = handler Object.onReco = GetRef("handler");

#### Event Object Info:

Bubbles	No
To invoke	User presses the escape key or the number of key strokes meets specified value.
Default action	Returns the key being pressed

### Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

#### 5 4.4.3 onTimeout

Fires when no phrase finish event is received before time out. The event halts the recognition process automatically.

10

### Syntax:

Inline HTML	<DTMF onTimeout="handler" ...>
Event property (in ECMAScript)	Object.onTimeout = <i>handler</i> Object.onTimeout = GetRef("handler");

### Event Object Info:

Bubbles	No
To invoke	No DTMF key stroke is detected within the timeout specified.
Default action	None

### Event Properties:

15

Although the event handler does not receive properties directly, the handler can query the event object for data.

## 5 CallControl Object

---

Represents the telephone interface (call, terminal, and connection) of the telephone voice browser. This  
5 object is as native as window object in a GUI browser. As such, the lifetime of the telephone object is the same as the browser instance itself. A voice browser for telephony instantiates the telephone object, one for each call. Users don't instantiate or dispose the  
10 object.

At this point, only features related to first-party call controls are exposed through this object.

### 5.1 Properties

15

- **address**

Read-only. XML DOM node object. Implementation specific. This is the address of the caller. For PSTN, may a combination of ANI and ALI. For VoIP,  
20 this is the caller's IP address.

- **ringsBeforeAnswer**

Number of rings before answering an incoming call. Default is infinite, meaning the developer  
25 must specifically use the Answer( ) method below to answer the phone call. When the call center uses ACD to queue up the incoming phone calls, this number can be set to 0.

## 5.2 Methods

Note: all the methods here are synchronous.

### 5.2.1 Transfer

5 Transfers the call. For a blind transfer, the system may terminate the original call and free system resources once the transfer completes.

#### Syntax:

10 telephone.Transfer(strText);

#### Parameters:

- o **strText:** Required. The address of the intended receiver.

#### Return value:

15 None.

#### Exception:

Throws an exception when the call transfer fails. e.g., when end party is busy, no such number, fax or answering machine answers.

### 20 5.2.2 Bridge

25 Third party transfer. After the call is transferred, the browser may release resources allocated for the call. It is up to the application to recover the session state when the transferred call returns using strUID. The underlying telephony platform may route the returning call to a different browser. The call

can return only when the recipient terminates the call.

**Syntax:**

5           telephone.Bridge(strText, strUID, [imaxTime]  
);

**Parameters:**

- 10           o **strText:** Required. The address of the  
intended receiver.
- o **strUID:** Required. The session ID uniquely  
identifying the current call. When the  
transferred call is routed back, the strUID  
will appear in the address attribute.
- 15           o **imaxTime:** Optional. Maximum duration in  
seconds of the transferred call. If  
unspecified, defaults to platform-internal  
value

**Return value:**

20           None.

**Exception:**

None.

**5.2.3           Answer**

25           Answers the phone call.

**Syntax:**

          telephone.Answer( );

**Return value:**

None.

**Exception:**

Throws an exception when there is no  
connection. No onAnswer event will be fired  
5 in this case.

**5.2.4 Hangup**

Terminates the phone call. Has no effect if no  
call currently in progress.

10 **Syntax:**

telephone.Hangup( );

**Return value:**

None.

**Exception:**

15 None.

**5.2.5 Connect**

Starts a first-party outbound phone call.

20 **Syntax:**

telephone.Connect(strText, [iTimeout] );

**Parameters:**

- o **strText:** Required. The address of the  
intended receiver.
- 25 o **iTimeout:** Optional. The time in milliseconds  
before abandoning the attempt. If

unspecified, defaults to platform-internal value.

**Return value:**

None.

5      **Exception:**

Throws an exception when the call cannot be completed, including encountering busy signals or reaching a FAX or answering machine (Note: hardware may not support this feature).

5.2.6      **Record**

Record user audio to file.

**Syntax:**

15      telephone.Record(url, endSilence,  
[maxTimeout], [initialTimeout]);

**Parameters:**

- o **url:** Required. The url of the recorded results.
- 20      o **endSilence:** Required. Time in milliseconds to stop recording after silence is detected.
- o **maxTimeout:** Optional. The maximum time in seconds for the recording. Default is platform-specific.
- 25      o **initialTimeout:** Optional. Maximum time (in milliseconds) of silence allowed at the beginning of a recording.

**Return value:**

None.

**Exception:**

Throws an exception when the recording can  
5 not be written to the url.

**5.3 Event Handlers**

App developers using telephone voice browser may implement the following event handlers.

**5.3.1 onIncoming( )**

10 Called when the voice browser receives an incoming phone call. All developers can use this handler to read caller's address and invoke customized features before answering the phone call.

15

**5.3.2 onAnswer( )**

Called when the voice browser answers an incoming phone call.

20 **5.3.3 onHangup( )**

Called when user hangs up the phone. This event is NOT automatically fired when the program calls the Hangup or Transfer methods.



#### 5.4 Example

This example shows scripting wired to the call control events to manipulate the telephony session.

```
<HTML>
<HEAD>
  <TITLE>Logon Page</TITLE>
</HEAD>
  <SCRIPT>
10   var focus;
      function RunSpeech() {
          if (logon.user.value == "") {
              focus="user";
              p_uid.Start(); g_login.Start();
15   dtmf.Start(); return;
          }
          if (logon.pass.value == "") {
              focus="pin";
              p_pin.Start(); g_login.Start();
20   dtmf.Start(); return;
          }
          p_thank.Start(); logon.submit();
      }
      function login_reco() {
25   res = event.srcElement.recoResult;
      pNode = res.selectSingleNode("//uid");
      if (pNode != null)
          logon.user.value = pNode.xml;
      pNode = res.selectSingleNode("//password");
30   if (pNode != null)
          logon.pass.value = pNode.xml;
      }
      function dtmf_reco() {
          res = event.srcElement.dtmfResult;
35   if (focus == "user")
              logon.user.value = res;
          else
              logon.pin.value = res;
      }
40  </SCRIPT>
  <SCRIPT for="callControl" event="onIncoming">
    <!--
```

```
        // read address, prepare customized stuff if
    any
        callControl.Answer();
    //
5    </SCRIPT>
    <SCRIPT for="callControl" event="onOffhook">
        <!--
            p_main.Start(); g_login.Start(); dtmf.Start();
            focus="user";
10    //
        </SCRIPT>
        <SCRIPT for="window" event="onload">
            <!--
                if (logon.user.value != "") {
15                    p_retry.Start();
                    logon.user.value = "";
                    logon.pass.value = "";
                    checkFields();
                }
20    //
        </SCRIPT>
    <BODY>
        <reco id="g_login"
            onReco="login_reco(); runSpeech()"
25            timeout="5000"
            onTimeout="p_miss.Start(); RunSpeech()" >
            <grammar
                src=http://kokanee1/etradedemo/speechonly/login.xml/>
            </ reco >
30    <dtmf id="dtmf"
            escape="#"
            onkeypress="g_login.Stop();"
            onReco="dtmf_reco();RunSpeech()"
            interdigitTimeout="5000"
35            onTimeout="dtmf.Flush();
                p_miss.Start();RunSpeech()" />

        <prompt id="p_main">Please say your user I D and pin
            number</prompt>
40    <prompt id="p_uid">Please just say your user I
            D</prompt>
        <prompt id="p_pin">Please just say your pin
            number</prompt>
```

```
<prompt id="p_miss">Sorry, I missed that</prompt>
<prompt id="p_thank">Thank you. Please wait while I
  verify your identity</prompt>
<prompt id="p_retry">Sorry, your user I D and pin
5  number do not match</prompt>
```

```
<H2>Login</H2>
<form id="logon">
  UID:  <input name="user" type="text"
10      onChange="runSpeech()" />
  PIN:  <input name="pass" type="password"
        onChange="RunSpeech()" />
</form>
</BODY>
15 </HTML>
```

## 6 Controlling dialog flow

---

### 6.1 Using HTML and script to implement dialog flow

20 This example shows how to implement a simple dialog flow which seeks values for input boxes and offers context-sensitive help for the input. It uses the title attribute on the HTML input mechanisms (used in a visual browser as a "tooltip" mechanism) to help  
25 form the content of the help prompt.

```
<html>
  <title>Context Sensitive Help</title>
<head>
30 <script>  var focus;
            function RunSpeech() {
              if (trade.stock.value == "") {
                focus="trade.stock";
                p_stock.Start();
35          return;
            }
```

```

    }
    if (trade.op.value == "") {
        focus="trade.op";
        p_op.Start();
5       return;
    }
    //.. repeat above for all fields
    trade.submit();
}
10  function handle() {
        res = event.srcElement.recoResult;
        if (res.text == "help") {
            text = "Please just say";
            text += document.all[focus].title;
15         p_help.Start(text);
        } else {
            // proceed with value assignments
        }
    }
20  </script>
    </head>
    <body>
        <prompt id="p_help" onComplete="checkFields()" />
        <prompt id="p_stock"
25  onComplete="g_stock.Start()">Please say the stock
        name</prompt>
        <prompt id="p_op" onComplete="g_op.Start()">Do you
        want to buy or sell</prompt>
        <prompt id="p_quantity"
30  onComplete="g_quantity.Start()">How many
        shares?</prompt>
        <prompt id="p_price"
        onComplete="g_price.Start()">What's the price</prompt>
35  <reco id="g_stock" onReco="handle(); checkFields()" >
        <grammar src="./g_stock.xml" />
        </ reco >

        <reco id="g_op" onReco="handle(); checkFields()" />
40  <grammar src="./g_op.xml" />
        </ reco >

```

```
<reco id="g_quantity" onReco="handle(); checkFields()"
/>
    <grammar src="./g_quant.xml" />
</ reco >
5
<reco id="g_price" onReco="handle(); checkFields()" />
    <grammar src="./g_quant.xml" />
</ reco >

10
<form id="trade">
    <input name="stock" title="stock name" />
    <select name="op" title="buy or sell">
        <option value="buy" />
15        <option value="sell" />
    </select>
    <input name="quantity" title="number of shares"
/>
    <input name="price" title="price" />
20 </form>
</body>
</html>
```

## 6.2 Using SMIL

The following example shows activation of prompt and  
25 reco elements using SMIL mechanisms.

```
<html xmlns:t="urn:schemas-microsoft-com:time"
      xmlns:sp="urn:schemas-microsoft-
com:speech">
30 <head>
    <style>
        .time { behavior: url(#default#time2); }
    </style>
    </head>
35 <body>

    <input name="txtBoxOrigin" type="text"/>
    <input name="txtBoxDest" type="text" />

40 <sp:prompt class="time" t:begin="0">
    Please say the origin and destination cities
```

```
</sp:prompt>

<t:par t:begin="time.end"
t:repeatCount="indefinitely"
5   <sp:reco class="time" >
      <grammar src="./city.xml" />
      <bind targetElement="textBoxOrigin"
        value="//origin_city" />
      <bind targetElement="textBoxDest"
10   test="/sml/dest_city[@confidence $gt$ 40]"
        value="//dest_city" />
      </sp:reco>
    </t:par>

15  </body>
    </html>
```

2025 RELEASE UNDER E.O. 14176

## APPENDIX B

## 1 QA speech control

The QA control adds speech functionality to the primary control to which it is attached. Its object model is an abstraction of the content model of the exemplary tags in Appendix A.

## 1.1 QA control

```

    <Speech:QA
      id="..."
      controlsToSpeechEnable="..."
      speechIndex="..."
      ClientTest="..."
      runat="server" >

      <Question ...>

      <Statement ...>
      ...
      <Answer ...>

      <Confirm ...>
      ...
      <Command ...>
      ...
    </Speech:QA>
  
```

## 1.1.1 Core properties

string ControlsToSpeechEnable

ControlsToSpeechEnable specifies the list of IDs of the primary controls to speech enable. IDs are comma delimited.

## 1.1.2 Activation mechanisms

int SpeechIndex

SpeechIndex specifies the ordering information of the QA control - this is used by RunSpeech. Note: If more than one QA control has the same SpeechIndex, RunSpeech will execute them in source order. In situations where some QA controls

have SpeechIndex specified and some QA controls do not, RunSpeech will order the QA controls first by SpeechIndex, then by source order.

## 5 string ClientTest

ClientTest specifies a client-side script function which returns a boolean value to determine when the QA control is considered available for selection by the RunSpeech algorithm. The system strategy can therefore be changed by using this as a condition to activate or de-activate QA controls more sensitively than SpeechIndex. If not specified, the QA control is considered available for activation.

### 1.1.3 Questions, Statements, Answers, Confirms and Commands

#### Question[] Questions

QA control contains an array of question objects or controls, defined by the dialog author. Each question control will typically relate to a function of the system, eg asking for a value, etc. Each question control may specify an activation function using the ClientTest attribute, so an active QA control may ask different kinds of questions about its primary control under different circumstances. For example, the activation condition for main question Q\_Main may be that the corresponding primary control has no value, and the activation condition for a Q\_GiveHelp may be that the user has just requested help. Each Question may specify answer controls from within the QA control which are activated when the question control is outputted.



Statement[] Statement

QA control contains an array of statement objects or controls. Statements are used to provide information to the listener, such as welcome prompts.

5

Answer[] Answers

QA control contains an array of answer objects or controls. An answer control is activated directly by a question control within the QA control, or by a StartEvent from the Primary control. Where multiple answers are used, they will typically reflect answers to the system functions, e.g. A\_Main might provide a value in response to Q\_Main, and A\_Confirm might providing a yes/no + correction to Confirm.

10

15 Confirm[] Confirm

QA control may contain a confirm object or control. This object is a mechanism provided to the dialog authors which simplify the authoring of common confirmation subdialogs.

20 Command[] Command

A Command array holds a set of command controls. Command controls can be thought of as answer controls without question controls, whose behavior on recognition can be scoped down the control tree.

25 1.2 Question control

The question control is used for the speech output relating to a given primary control. It contains a set of prompts for presenting information or asking a question, and a list of ids of the answer controls, which may provide an answer to that question. If multiple answer controls are specified, these grammars are loaded in parallel when the

30

question is activated. An exception will be thrown if no answer control is specified in the question control.

```
<Question
```

```
5      id="..."
      ClientTest="..."
      Answers="..."
      Count="..."
      initialTimeout="..."
10     babbleTimeout="..."
      maxTimeout="..."
      Modal="..."
      PromptFunction="..."
      OnClientNoReco="..." >
```

```
15     <prompt ... />
```

```
    ...
```

```
</Question>
```

```
20 string ClientTest
```

ClientTest specifies the client-side script function returning a boolean value which determines under which circumstances a question control is considered active within its QA control (the QA control itself must be active for the question to be evaluated). For a given QA control, the first question control with a true condition is selected for output. For example, the function may be used to determine whether to output a question which asks for a value ("Which city do you want?") or which attempts to confirm it ("Did you say London?"). If not specified, the question condition is considered true.

## Prompt[] Prompts

The prompt array specifies a list of prompt objects, discussed below. Prompts are also able to specify conditions of selection (via client functions), and during  
 5 RunSpeech execution only the first prompt with a true condition is selected for playback.

## String Answers

Answers is an array of references by ID to controls that  
 10 are possible answers to the question. The behavior is to activate the grammar from each valid answer control in response to the prompt asked by the question control.

## Integer initialTimeout

15 The time in milliseconds between start of recognition and the detection of .speech. This value is passed to the recognition platform, and if exceeded, an onSilence event will be thrown from the recognition platform. If not specified, the speech platform will use a default value.

## Integer babbleTimeout

The period of time in milliseconds in which the recognition server or other recognizer must return a result after detection of speech. For recos in "tap-and-talk" scenarios  
 25 this applies to the period between speech detection and the recognition result becoming available. For recos in dictation scenarios, this timeout applies to the period between speech detection and each recognition return - i.e. the period is restarted after each return of results or  
 30 other event. If exceeded, the onClientNoReco event is thrown but different status codes are possible. If there has been any kind of recognition platform error that is

detectable and the babbleTimeout period has elapsed, then an onClientNoReco is thrown but with a status code -3. Otherwise if the recognizer is still processing audio - e.g. in the case of an exceptionally long utterance or if the user has kept the pen down for an excessive amount of time - the onClientNoReco event is thrown, with status code -15. If babbleTimeout is not specified, the speech platform will default to an internal value.

## 10 Integer maxTimeout

The period of time in milliseconds between recognition start and results returned to the client device browser. If exceeded, the onMaxTimeout event is thrown by the browser - this caters for network or recognizer failure in distributed environments. For recos in dictation scenarios, as with babbleTimeout, the period is restarted after the return of each recognition or other event. Note that the maxTimeout attribute should be greater than or equal to the sum of initialTimeout and babbleTimeout. If not specified, the value will be a browser default.

## bool modal

When modal is set to true, no answers except the immediate set of answers to the question are activated (i.e. no scoped Answers are considered). The default is false. For Example, this attribute allows the application developer to force the user of the client device to answer a particular question.

## 30 String PromptFunction( prompt)

PromptFunction specifies a client-side function that will be called once the question has been selected but before

the prompt is played. This gives a chance to the application developer to perform last minute modifications to the prompt that may be required. PromptFunction takes the ID of the target prompt as a required parameter.

5

string OnClientNoReco

OnClientNoReco specifies the name of the client-side function to call when the NoReco (mumble) event is received.

#### 10 1.2.1 Prompt Object

The prompt object contains information on how to play prompts. All the properties defined are read/write properties.

```

15     <prompt
        id="..."
        count="..."
        ClientTest="..."
        source="..."
        bargeIn="..."
20     onClientBargein="..."
        onClientComplete="..."
        onClientBookmark="..." >
        ...text/markup of the prompt...
    </prompt>

```

25

int count

Count specifies an integer which is used for prompt selection. When the value of the count specified on a prompt matches the value of the count of its question control, the prompt is selected for playback. Legal values are 0 - 100.

30

```

    <Question id=Q_Ask">
        <prompt count="1"> Hello </prompt>
35     <prompt count="2"> Hello again </prompt>
    </Question>

```

In the example, when `Q_Ask.count` is equal to 1, the first prompt is played, and if it is equal to 2 (i.e. the question has already been output before), the second prompt is then played.

`string ClientTest`

`ClientTest` specifies the client-side script function returning a boolean value which determines under which circumstances a prompt within an active question control will be selected for output. For a given question control, the first prompt with a true condition is selected. For example, the function may be used to implement prompt tapering, eg ("Which city would you like to depart from?" for a function returning true if the user is a first-timer, or "Which city?" for an old hand). If not specified, the prompt's condition is considered true.

`string InlinePrompt`

The prompt property contains the text of the prompt to play. This is defined as the content of the prompt element. It may contain further markup, as in TTS rendering information, or `<value>` elements. As with all parts of the page, it may also be specified as script code within `<script>` tags, for dynamic rendering of prompt output.

`string Source`

`Source` specifies the URL from which to retrieve the text of the prompt to play. If an inline prompt is specified, this property is ignored.

Bool BargeIn

BargeIn is used to specify whether or not barge-in (wherein the user of the client device begins speaking when a prompt is being played) is allowed on the prompt. The defaults is  
5 true.

string onClientBargein

onClientBargein specifies the client-side script function which is invoked by the bargein event.

10

string onClientComplete

onClientComplete specifies the client-side script function which is invoked when the playing of the prompt has competed.

15

string OnClientBookmark

OnClientBookmark accesses the name of the client-side function to call when a bookmark is encountered.

### 1.2.2 Prompt selection

20 On execution by RunSpeech, a QA control selects its prompt in the following way:

ClientTest and the count attribute of each prompt are evaluated in order. The first prompt with both ClientTest and count true is played. A missing count is considered  
25 true. A missing ClientTest is considered true.

### 1.3 Statement Control

Statement controls are used for information-giving system output when the activation of grammars is not required. This is common in voice-only dialogs. Statements are played  
30 only once per page if the playOnce attribute is true.

```

    <Statement
      id="..."
      playOnce="..."
      ClientTest="..."
      PromptFunction="..." >
      <prompt ... />
      ...
    </Statement >

```

bool playOnce

The playOnce attribute specifies whether or not a statement control may be activated more than once per page. playOnce is a Boolean attribute with a default (if not specified) of TRUE, i.e., the statement control is executed only once. For example, the playOnce attribute may be used on statement controls whose purpose is to output email messages to the end user. Setting playOnce="False" will provide dialog authors with the capability to enable a "repeat" functionality on a page that reads email messages.

string ClientTest

ClientTest specifies the client-side script function returning a boolean value which determines under which circumstances a statement control will be selected for output. RunSpeech will activate the first Statement with ClientTest equal to true. If not specified, the ClientTest condition is considered true.

String PromptFunction

PromptFunction specifies a client-side function that will be called once the statement control has been selected but before the prompt is played. This gives a chance to the authors to do last minute modifications to the prompt that may be required.



Prompt[] Prompt

The prompt array specifies a list of prompt objects. Prompts are also able to specify conditions of selection (via client functions), and during RunSpeech execution only the first prompt with a true condition is selected for playback.

```

10      <Speech:QA
        id="QA_Welcome"
        ControlsToSpeechEnable="Label1"
        runat="server" >

        <Statement id="WelcomePrompt" >
15          <prompt bargeIn="False"> Welcome </prompt>
        </Statement>
    </Speech:QA>

```

#### 1.4 Confirm Control

Confirm controls are special types of question controls. They may hold all the properties and objects of other questions controls, but they are activated differently. The RunSpeech algorithm will check the confidence score found in the confirmThreshold of the answer control of the ControlsToSpeechEnable. If it is too low, the confirm control is activated. If the confidence score of the answer control is below the confirmThreshold, then the binding is done but the onClientReco method is not called. The dialog author may specify more than one confirm control per QA control. RunSpeech will determine which confirm control to activate based on the function specified by ClientTest.

```

    <Answer ConfirmThreshold=... />
    <Confirm>
35      ...all attributes and objects of Question...

```

</Confirm>

### 1.5 Answer control

The answer control is used to specify speech input resources and features. It contains a set of grammars related to the primary control. Note that an answer may be used independently of a question, in multimodal applications without prompts, for example, or in telephony applications where user initiative may be enabled by extra-answers. Answer controls are activated directly by question controls, by a triggering event, or by virtue of explicit scope. An exception will be thrown if no grammar object is specified in the answer control.

```

15      <Answer
          id="..."
          scope="..."
          StartEvent="..."
          StopEvent="..."
          ClientTest="..."
20      onClientReco="..."
          onClientDTMF="..."
          autobind="..."
          server="..."
          ConfirmThreshold="..."
25      RejectThreshold="..." >

          <grammar ... />
          <grammar ... />
          ...
30      <dtmf ... />
          <dtmf ... />
          ...
          <bind ... />
          <bind ... />
35      ...
    </Answer>

```

string Scope

Scope holds the id of any named element on the page. Scope is used in answer control for scoping the availability of user initiative (mixed task initiative: i.e. service jump digressions) grammars. If scope is specified in an answer control, then it will be activated whenever a QA control corresponding to a primary control within the subtree of the contextual control is activated.

10 string StartEvent

StartEvent specifies the name of the event from the primary control that will activate the answer control (start the Reco object). This will be typically used in multi-modal applications, eg onMouseDown, for tap-and-talk.

15

string StopEvent

StopEvent specifies the name of the event from the primary control that will de-activate the answer control (stop the Reco object). This will be typically used in multi-modal applications, eg onMouseUp, for tap-and-talk.

20

string ClientTest

ClientTest specifies the client-side script function returning a boolean value which determines under which circumstances an answer control otherwise selected by scope or by a question control will be considered active. For example, the test could be used during confirmation for a 'correction' answer control to disable itself when activated by a question control, but mixed initiative is not desired (leaving only accept/deny answers controls active). Or a scoped answer control which permits a service jump can determine more flexible means of activation by

30

specifying a test which is true or false depending on another part of the dialog. If not specified, the answer control's condition is considered true.

## 5 Grammar[] Grammars

Grammars accesses a list of grammar objects.

## DTMF[] DTMFs

DTMFs holds an array of DTMF objects.

10

## Bind[] Binds

Binds holds a list of the bind objects necessary to map the answer control grammar results (dtmf or spoken) into control values. All binds specified for an answer will be executed when the relevant output is recognized. If no bind is specified, the SML output returned by recognition will be bound to the control specified in the ControlsToSpeechEnable of the QA control

15

## 20 string OnClientReco

OnClientReco specifies the name of the client-side function to call when spoken recognition results become available.

## string OnClientDTMF

25 OnClientDTMF holds the name of the client-side function to call when DTMF recognition results become available.

## boolean autobind

30

The value of autobind determines whether or not the system default bindings are implemented for a recognition return from the answer control. If unspecified, the default is

true. Setting autobind to false is an instruction to the system not to perform the automatic binding.

string server

- 5 The server attribute is an optional attribute specifying the URI of the speech server to perform the recognition. This attribute over-rides the URI of the global speech server attribute.

10 integer ConfirmThreshold

Holds a value representing the confidence level below which a confirm control question will be automatically triggered immediately after an answer is recognized within the QA control. Legal values are 0-100.

15

Note that where bind statements and onClientReco scripts are both specified, the semantics of the resulting Tags are that binds are implemented before the script specified in onClientReco.

20

integer RejectThreshold

- RejectThreshold specifies the minimum confidence score to consider returning a recognized utterance. If overall confidence is below this level, a NoReco event will be  
25 thrown. Legal values are 0-100.

#### 1.5.1 Grammar

- The grammar object contains information on the selection and content of grammars, and the means for processing recognition results. All the properties defined are  
30 read/write properties.

```

<Grammar
    ClientTest="..."
    Source="..."
    >
5    ...grammar rules...
</Grammar>

```

string ClientTest

The ClientTest property references a client-side boolean  
 10 function which determines under which conditions a grammar  
 is active. If multiple grammars are specified within an  
 answer control (e.g. to implement a system/mixed initiative  
 strategy, or to reduce the perplexity of possible answers  
 when the dialog is going badly), only the first grammar  
 15 with a true ClientTest function will be selected for  
 activation during RunSpeech execution. If this property is  
 unspecified, true is assumed.

string Source

20 Source accesses the URI of the grammar to load, if  
 specified.

string InlineGrammar

InlineGrammar accesses the text of the grammar if specified  
 25 inline. If that property is not empty, the Source attribute  
 is ignored.

### 1.5.2 Bind

The object model for bind follows closely its counterpart  
 client side tags. Binds may be specified both for spoken  
 30 grammar and for DTMF recognition returns in a single answer  
 control.

```

    <bind
      Value="..."
      TargetElement="..."
5      TargetAttribute="..."
      Test="..."
    />

```

string Value

10 Value specifies the text that will be bound into the target element. It is specified as an XPath on the SML output from recognition.

string TargetElement

15 TargetElement specifies the id of the primary control to which the bind statement applies. If not specified, this is assumed to be the ControlsToSpeechEnable of the relevant QA control.

20 string TargetAttribute

TargetAttribute specifies the attribute on the TargetElement control in which bind the value. If not specified, this is assumed to be the Text property of the target element.

25 string Test

The Test attribute specifies a condition which must evaluate to true on the binding mechanism. This is specified as an XML Pattern on the SML output from

30 recognition.

### 1.5.2.1 Automatic binding

The default behavior on the recognition return to a speech-enabled primary control is to bind certain properties into that primary control. This is useful for the dialog controls to examine the recognition results from the primary controls across turns (and even pages). Answer controls will perform the following actions upon receiving recognition results:

1. bind the SML output tree into the SML attribute of the primary control
2. bind the text of the utterance into the SpokenText attribute of the primary control
3. bind the confidence score returned by the recognizer into the Confidence attribute of the primary control.

Unless autobind="False" attribute is specified on an answer control, the answer control will perform the following actions on the primary control:

1. bind the SML output tree into the SML attribute;
2. bind the text of the utterance into the SpokenText attribute;
3. bind the confidence score returned by the recognizer into the Confidence attribute;

Any values already held in the attribute will be overwritten. Automatic binding occurs before any author-specified bind commands, and hence before any onClientReco script (which may also bind to these properties).



## 1.5.3 DTMF

DTMF may be used by answer controls in telephony applications. The DTMF object essentially applies a different modality of grammar (a keypad input grammar rather than a speech input grammar) to the same answer. The DTMF content model closely matches that of the client side output Tags DTMF element. Binding mechanisms for DTMF returns are specified using the targetAttribute attribute of DTMF object.

```

10      <DTMF
          firstTimeOut="..."
          interDigitTimeOut="..."
          numDigits="..."
15      flush="..."
          escape="..."
          targetAttribute="..."
          ClientTest="...">
20      <dtmfGrammar ...>
    </DTMF>

```

integer firstTimeOut

The number of milliseconds to wait between activation and the first key press before raising a timeout event.

integer interDigitTimeOut

The number of milliseconds to wait between key presses before raising a timeout event.

int numDigits

The maximum number of key inputs permitted during DTMF recognition.

bool flush

A flag which states whether or not to flush the telephony server's DTMF buffer before recognition begins. Setting flush to false permits DTMF key input to be stored between  
 5 recognition/page calls, which permits the user to 'type-ahead'.

string escape

Holds the string value of the key which will be used to end  
 10 DTMF recognition (eg '#').

string targetAttribute

TargetAttribute specifies the property on the primary control in which to bind the value. If not specified, this  
 15 is assumed to be the Text property of the primary control.

string ClientTest

The ClientTest property references a client-side boolean function which determines under which conditions a DTMF  
 20 grammar is active. If multiple grammars are specified within a DTMF object, only the first grammar with a true ClientTest function will be selected for activation during RunSpeech execution. If this property is unspecified, true is assumed.

#### 25 1.5.4 DTMFGrammar

DTMFGrammar maps a key to an output value associated with the key. The following sample shows how to map the "1" and "2" keys to text output values.

```
30 <dtmfgrammar>
    <key value="1">Seattle</key>
    <key value="2">Boston</key>
```

</dtmfgrammar>

## 1.6 Command control

The command control is a special variation of answer control which can be defined in any QA control. Command controls are forms of user input which are not answers to the question at hand (eg, Help, Repeat, Cancel), and which do not need to bind recognition results into primary controls. If the QA control specifies an activation scope, the command grammar is active for every QA control within that scope. Hence a command does not need to be activated directly by a question control or an event, and its grammars are activated in parallel independently of answer controls building process. Command controls of the same type at QA controls lower in scope can override superior commands with context-sensitive behavior (and even different / extended grammars if necessary).

```

<Command
  id="..."
  scope="..."
  type="..."
  RejectThreshold="..."
  onClientReco="..." >

  <Grammar ...>
  <dtmf ... >
  ...
</Command>

```

## string Scope

Scope holds the id of a primary control. Scope is used in command controls for scoping the availability of the command grammars. If scope is specified for a command control, the command's grammars will be activated whenever

a QA control corresponding to a primary control within the subtree of the contextual control is activated.

string type

- 5 Type specifies the type of command (eg 'help', 'cancel' etc.) in order to allow the overriding of identically typed commands at lower levels of the scope tree. Any string value is possible in this attribute, so it is up to the author to ensure that types are used correctly.

10

integer RejectThreshold

RejectThreshold specifies the minimum confidence level of recognition that is necessary to trigger the command in recognition (this is likely to be used when higher than  
15 usual confidence is required, eg before executing the result of a 'Cancel' command). Legal values are 0-100.

string onClientReco

- 20 onCommand specifies the client-side script function to execute on recognition of the command control's grammar.

Grammar Grammar

The grammar object which will listen for the command.

- 25 DTMF DTMF

The dtmf object which will activate the command.

## 2 Types of Initiatives and Dialog Flows

---

Using the control described above, various forms of initiatives can be developed, some examples are provided below:

### 2.1 Mixed initiative Dialogs

Mixed initiative dialogs provide the capability of accepting input for multiple controls with the asking of a single question. For example, the answer to the question "what are your travel plans" may provide values for an origin city textbox control, a destination city textbox control and a calendar control ("Fly from Puyallup to Yakima on September 30<sup>th</sup>").

A robust way to encode mixed initiative dialogs is to handwrite the mixed initiative grammar and relevant binding statements, and apply these to a single control.

The following example shows a single page used for a simple mixed initiative voice interaction about travel. The first QA control specifies the mixed initiative grammar and binding, and a relevant prompt asking for two items. The second and third QA controls are not mixed initiative, and so bind directly to their respective primary control by default (so no bind statements are required). The RunSpeech algorithm will select the QA controls based on an attribute "SpeechIndex" and whether or not their primary controls hold valid values.

```
<%@ Page language="c#" AutoEventWireup="false"
inherits="SDN.Page" %>
```

```
<%@ Register tagPrefix="SDN" Namespace="SDN" Assembly="SDN"
%>
<html>
```

```
5 <body>
```

```
<Form id="WebForm1" method=post runat="server">
```

```
<ASP:Label id="Label1" runat="server">Departure
city</ASP:Label>
```

```
<ASP:TextBox id="TextBox1" runat="server" />
```

```
10 <br>
```

```
<ASP:Label id="Label2" runat="server">Arrival
city</ASP:Label>
```

```
<ASP:TextBox id="TextBox2" textchanged="TextChanged"
runat="server" />
```

```
15 <!--speech information -->
```

```
<Speech:QA id="QAmixed" controlsToSpeechEnable="TextBox1"
speechIndex="1" runat="server">
```

```
20 <Question id="Q1" Answers="A1">
```

```
<prompt>"Please say the cities you want to fly
from and to"</prompt>
```

```
</Question>
```

```
25 <Answer id="A1" >
```

```
<grammar src="..." />
```

```
<bind targetElement="TextBox1"
value="/sml/path1"/>
```

```
<bind targetElement="TextBox2"
30 value="/sml/path2"/>
```

```
</Answer>
```

```
</Speech:QA>
```

```
35 <Speech:QA id="QA1" controlsToSpeechEnable="TextBox1"
speechIndex="2" runat="server">
```

```
<Question id="Q1" Answers="A1">
```

```
<prompt>"What's the departure city?"</prompt>
40 </Question>
```

```
<Answer id="A1">
```

```
<grammar src="..." />
```

```
</Answer>
```

```
</Speech:QA>
```

```
45 <Speech:QA id="QA2" controlsToSpeechEnable="TextBox2"
speechIndex="3" runat="server">
```

```

<Question id="Q1" Answer="A1">
  <prompt>"What's the arrival city"</prompt>
</Question>

```

```

5    <Answer id="A1" >
      <grammar src="..." />
    </Answer>
</Speech:QA>

```

```

10  </Form>
    </body>
    </html>

```

## 2.2 Complex Mixed Initiative

Application developers can specify several answer to the same question control with different levels of initiatives. Conditions are specified that will select one of the answers when the question is asked, depending on the initiative settings that they require. An example is provided below:

```

20  <Speech:QA
      id="QA_Panel2"
      ControlsToSpeechEnable="Panel2"
      runat="server" >
25      <Question                                answers="systemInitiative,
mixedInitiative" .../>
      <Answer id="systemInitiative"
30          ClientTest="systemInitiativeCond"
          onClientReco="SimpleUpdate" >
          <grammar src="systemInitiative.gram" />
      </Answer>
      <Answer id="mixedInitiative"
35          ClientTest="mixedInitiativeCond"
          onClientReco="MixedUpdate" >
          <grammar src="mixedInitiative.gram" />
      </Answer>
    </Speech:QA>

```

40 Application developers can also specify several question controls in a QA control. Some question controls can allow

a mixed initiative style of answer, whilst others are more directed. By authoring conditions on these question controls, application developer can select between the questions depending on the dialogue situation.

5

In the following example the mixed initiative question asks the value of the two textboxes at the same time (e.g., 'what are your travel plans?') and calls the mixed initiative answer (e.g., 'from London to Seattle'). If this  
10 fails, then the value of each textbox is asked separately (e.g., 'where do you leave from' and 'where are you going to') but, depending on the conditions, the mixed-initiative grammar may still be activated, thus allowing users to provide both values.

15

```
<Speech:QA
  id="QA_Panel2"
  ControlsToSpeechEnable="TextBox1, TextBox2"
  runat="server" >
```

20

```
<Question
  ClientTest="AllEmpty()"
  answers="AnsAll"
  .../>
```

25

```
<Question
  ClientTest="TextBox1IsEmpty()"
  answers="AnsAll, AnsTextBox1" .../>
```

30

```
<Question
  ClientTest="TextBox2IsEmpty()"
  answers="AnsAll, AnsTextBox2" .../>
```

35

```
<Answer
  id="AnsTextBox1"
  onClientReco="SimpleUpdate">
    <grammar src="AnsTextBox1.gram" />
  </Answer>
```

40

```
<Answer
  id="AnsTextBox2"
  onClientReco="SimpleUpdate" >
    <grammar src=" AnsTextBox2.gram" />
```



```

        </Answer>
        <Answer
            id="AnsAll"
            ClientTest="IsMixedInitAllowed()"
5            onClientReco="MixedUpdate"
            >
            <grammar src="AnsAll.gram" />
        </Answer>
    </Speech:QA>

```

## 10 2.3 User initiative

Similar to the command control, a standard QA control can specify a scope for the activation of its grammars. Like a command control, this QA control will activate the grammar from a relevant answer control *whenever another QA control* 15 *is activated within the scope of this context*. Note that its question control will only be asked if the QA control itself is activated.

```

    <Speech:QA
        id="QA_Panel2"
        ControlsToSpeechEnable="Panel2"
        runat="server" >

        <Question ... />
        <Answer id="AnswerPanel2"
25             scope="Panel2"
                onClientReco="UpdatePanel2()" >
            <grammar src="Panel2.gram" />
        </Answer>
    </Speech:QA>
30

```

This is useful for dialogs which allow 'service jumping' - user responses about some part of the dialog which is not directly related to the question control at hand.

## 2.4 Short time-out confirms

Application developers can write a confirmation as usual but set a short time-out. In the timeout handler, code is provided to that accept the current value as exact.

```

5      <Speech:QA
        id="QA_Panel2"
        ControlsToSpeechEnable="Panel2"
        runat="server" >
10      <Confirm timeout="20"
onClientTimeout="AcceptConfirmation"... />
        <Answer id="CorrectPanel2"
            onClientReco="UpdatePanel2()" >
15      <grammar src="Panel2.gram" />
        </Answer>
    </Speech:QA>

```

## 2.5 Dynamic prompt building and editing

The promptFunction script is called after a question control is selected but before a prompt is chosen and played. This lets application developers build or modify the prompt at the last minute. In the example below, this is used to change the prompt depending on the level of experience of the users.

```

25      <script language=javascript>
        function GetPrompt() {
            if(experiencedUser == true)
                Prompt1.Text = "What service do you
30      want?";
            else
                Prompt1.Text = "Please choose between
                e-mail,
                calendar and news";
35      return;
        }
    </script>
    <Speech:QA
        id="QA_Panel2"

```

```

ControlsToSpeechEnable="Panel2"
runat="server" >

<Question PromptFunction="GetPrompt"... >
5      <Prompt id="Prompt1" />
    </Question>
    <Answer ... />
</Speech:QA>

```

## 2.6 Using semantic relationships

10 Recognition and use of semantic relationships can be done by studying the result of the recognizer inside the onReco event handler.

```

15 <script language="javascript">
    function Reco() {
        /*

```

Application developers can access the SML returned by the recogniser or recognition server. If a semantic relationship (like sport-news) is identified, the confidence of the individual elements can be increased or take any other appropriate action.

```

    */
    }
25 </script>
    <Speech:QA
        id="QA_Panel2"
        ControlsToSpeechEnable="Panel2"
        runat="server" >

30      <Question ... />
      <Answer onClientReco="Reco" >
          <grammar src="Panel2.gram" />
      </Answer>
35 </Speech:QA>

```

### 3 Implementation and Application of RunSpeech

---

A mechanism is needed to provide voice-only clients with the information necessary to properly render speech-enabled pages. Such a mechanism must provide the execution of dialog logic and maintain state of user prompting and grammar activation as specified by the application developer.

Such a mechanism is not needed for multimodal clients. In the multimodal case, the page containing speech-enabled controls is visible to the user of the client device. The user of the client device may provide speech input into any visible speech-enabled control in any desired order using the a multimodal paradigm.

The mechanism used by voice-only clients to render speech-enabled pages is the RunSpeech script or algorithm. The RunSpeech script relies upon the SpeechIndex attribute of the QA control and the SpeechGroup control discussed below.

#### 3.1 SpeechControl

During run time, the system parses a control script or webpage having the server controls and creates a tree structure of server controls. Normally the root of the tree is the Page control. If the control script uses custom or user control, the children tree of this custom or user control is expanded. Every node in the tree has an ID and it is easy to have name conflict in the tree when it expands. To deal with possible name conflict, the system includes a concept of NamingContainer. Any node in the tree

can implement NamingContainer and its children lives within that name space.

The QA controls can appear anywhere in the server control tree. In order to easily deal with SpeechIndex and manage client side rendering, a SpeechGroup control is provided. The Speechgroup control is hidden from application developer.

One SpeechGroup control is created and logically attached to every NamingContainer node that contain QA controls in its children tree. QA and SpeechGroup controls are considered members of its direct NamingContainer's SpeechGroup. The top level SpeechGroup control is attached to the Page object. This membership logically constructs a tree - a logical speech tree - of QA controls and SpeechGroup controls.

For simple speech-enabled pages or script (i.e., pages that do not contain other NamingContainers), only the root SpeechGroup control is generated and placed in the page's server control tree before the page is sent to the voice-only client. The SpeechGroup control maintains information regarding the number and rendering order of QA controls on the page.

For pages containing a combination of QA control(s) and NamingContainer(s), multiple SpeechGroup controls are generated: one SpeechGroup control for the page (as described above) and a SpeechGroup control for each NamingContainer. For a page containing NamingContainers, the page-level SpeechGroup control maintains QA control

information as described above as well as number and rendering order of composite controls. The SpeechGroup control associated with each NamingContainer maintains the number and rendering order of QAs within each composite.

5

The main job of the SpeechGroup control is to maintain the list of QA controls and SpeechGroups on each page and/or the list of QA controls comprising a composite control. When the client side markup script (e.g. HTML) is generated, each SpeechGroup writes out a QACollection object on the client side. A QACollection has a list of QA controls and QACollections. This corresponds to the logical server side speech tree. The RunSpeech script will query the page-level QACollection object for the next QA control to invoke during voice-only dialog processing.

15

The page level SpeechGroup control located on each page is also responsible for:

20

- Determining that the requesting client is a voice-only client; and
- Generating common script and supporting structures for all QA controls on each page.

25

When the first SpeechGroup control renders, it queries the System.Web.UI.Page.Request.Browser property for the browser string. This property is then passed to the RenderSpeechHTML and RenderSpeechScript methods for each QA control on the page. The QA control will then render for the appropriate client(multimodal or voice-only).

### 3.2 Creation of SpeechGroup controls

During server-side page loading, the onLoad event is sent to each control on the page. The page-level SpeechGroup control is created by the first QA control receiving the onLoad event. The creation of SpeechGroup controls is done in the following manner: (assume a page containing composite controls)

Every QA control will receive onLoad event from run time code. onLoad for a QA:

- Get the QA's NamingContainer N1
- Search for SpeechGroup in the N1's children
  - If already exists, register QA control with this SpeechGroup. onLoad returns.
  - If not found:
    - Create a new SpeechGroup G1, inserts it into the N1's children
    - If N1 is not Page, find N1's NamingContainer N2
    - Search for SpeechGroup in N2's children, if exists, say G2, add G1 to G2. If not, create a new one G2, inserts in to N2's children
    - Recursion until the NamingContainer is the Page (top level)

During server-side page rendering, the Render event is sent to the speech-enabled page. When the page-level SpeechGroup control receives the Render event, it generates client side script to include RunSpeech.js and inserts it into the page that is eventually sent to the client device. It also calls

all its direct children to render speech related HTML and scripts. If a child is SpeechGroup, the child in turn calls its children again. In this manner, the server rendering happens along the server side logical speech tree.

5

When a SpeechGroup renders, it lets its children (which can be either QA or SpeechGroup) render speech HTML and scripts in the order of their SpeechIndex. But a SpeechGroup is hidden and doesn't naturally have a SpeechIndex. In fact, a SpeechGroup will have the same SpeechIndex as its NamingContainer, the one it attaches to. The NamingContainer is usually a UserControl or other visible control, and an author can set SpeechIndex to it.

10

### 3.3 RunSpeech

15

The purpose of RunSpeech is to permit dialog flow via logic which is specified in script or logic on the client. In one embodiment, RunSpeech is specified in an external script file, and loaded by a single line generated by the server-side rendering of the SpeechGroup control, e.g.:

20

```
<script language="javascript"
src="/scripts/RunSpeech.js" />
```

25

The RunSpeech.js script file should expose a means for validating on the client that the script has loaded correctly and has the right version id, etc. The actual validation script will be automatically generated by the page class as inline functions that are executed after the attempt to load the file.

30

Linking to an external script is functionally equivalent to specifying it inline, yet it is both more efficient, since



browsers are able to cache the file, and cleaner, since the page is not cluttered with generic functions.

### 3.4 Events

#### 3.4.1 Event wiring

- 5 Tap-and-talk multimodality can be enabled by coordinating the activation of grammars with the onMouseDown event. The wiring script to do this will be generated by the Page based on the relationship between controls (as specified in the ControlsToSpeechEnable property of the QA control in).

10

For example, given an asp:TextBox and its companion QA control adding a grammar, the <input> and <reco> elements are output by each control's Render method. The wiring mechanism to add the grammar activation command is performed by client-side script generated by the Page, which changes the attribute of the primary control to add the activation command before any existing handler for the activation event:

15

20

```
<!-- Control output -->
<input id="TextBox1" type="text" .../>
<reco id="Reco1" ... />
    <grammar src="..." />
</reco>
```

25

```
<!-- Page output -->
<script>
    TextBox1.onMouseDown =
    "Reco1.Start();" + TextBox1.onMouseDown;
</script>
```

30

By default, hook up is via onMousedown and onMouseup events, but both StartEvent and StopEvent can be set by web page author.

The textbox output remains independent of this modification and the event is processed as normal if other handlers were present.

### 3.4.2 Page Class properties

- 5 The Page also contains the following properties which are available to the script at runtime:

SML - a name/value pair for the ID of the control and it's associated SML returned by recognition.

- 10 SpokenText - a name/value pair for the ID of the control and it's associated recognized utterance

Confidence - a name/value pair for the ID of the control and it's associated confidence returned by the recognizer.

## 15 4 RunSpeech Algorithm

---

- The RunSpeech algorithm is used to drive dialog flow on the client device. This may involve system prompting and dialog management (typically for voice-only dialogs), and/or processing of speech input (voice-only and multimodal  
20 dialogs). It is specified as a script file referenced by URI from every relevant speech-enabled page (equivalent to inline embedded script).

- Rendering of the page for voice only browsers is done in  
25 the following manner:

The RunSpeech module or function works as follows  
(RunSpeech is called in response to document.onreadystate becoming "complete"):

- (1) Find the first active QA control in speech index order (determining whether a QA control is active is explained below).
- (2) If there is no active QA control, submit the  
5 page.
- (3) Otherwise, run the QA control.

A QA control is considered active if and only if:

- 10 (1) The QA control's ClientTest either is not present or returns true, AND
- (2) The QA control contains an active question control or statement control (tested in source order), AND
- 15 (3) Either:
  - a. The QA control contains only statement controls, OR
  - b. At least one of the controls referenced by the QA control's ControlsToSpeechEnable has an  
20 empty or default value.

A question control is considered active if and only if:

- 25 (1) The question control's ClientTest either is not present or returns true, AND
- (2) The question control contains an active prompt object.

A prompt object is considered active if and only if:

- 30 (1) The prompt object's ClientTest either is not present or returns true, AND

- (2) The prompt object's Count is either not present, or is less than or equal to the Count of the parent question control.

5 A QA control is run as follows:

- (1) Determine which question control or statement control is active and increment its Count.
- (2) If a statement control is active, play the prompt and exit.
- (3) If a question control is active, play the prompt and start the Recos for each active answer control and command control.

15 An answer control is considered active if and only if:

- (1) The answer control's ClientTest either is not present or returns true, AND
- (2) Either:
  - a. The answer control was referenced in the active question control's Answers string, OR
  - b. The answer control is in Scope

A command control is considered active if and only if:

- (1) It is in Scope, AND
- (2) There is not another command control of the same Type lower in the scope tree.

30 RunSpeech relies on events to continue driving the dialog - as described so far it would stop after running a single QA control. Event handlers are included for

Prompt.OnComplete, Reco.OnReco, Reco.OnSilence, Reco.OnMaxTimeout, and Reco.OnNoReco. Each of these will be described in turn.

5 RunSpeechOnComplete works as follows:

- (1) If the active Prompt object has an OnClientComplete function specified, it is called.
- (2) If the active Prompt object was contained within  
10 a statement control, or a question control which had no active answer controls, RunSpeech is called.

RunSpeechOnReco works as follows:

- 15 (1) Some default binding happens - the SML tree is bound to the SML attribute and the text is bound to the SpokenText attribute of each control in ControlsToSpeechEnable.
- (2) If the confidence value of the recognition result  
20 is below the ConfidenceThreshold of the active answer control, the Confirmation logic is run.
- (3) Otherwise, if the active answer control has on OnClientReco function specified, it is called, and then RunSpeech is called.

25 RunSpeechOnReco is responsible for creating and setting the SML, SpokenText and Confidence properties of the ControlsToSpeechEnable. The SML, SpokenText and Confidence properties are then available to scripts at runtime.

30 RunSpeechOnSilence, RunSpeechOnMaxTimeout, and RunSpeechOnNoReco all work the same way:

- (1) The appropriate OnClientXXX function is called, if specified.
- (2) RunSpeech is called.

5

Finally, the Confirmation logic works as follows:

- (1) If the parent QA control of the active answer control contains any confirm controls, the first active confirm control is found (the activation of a confirm control is determined in exactly the same way as the activation of a question control).
- (2) If no active confirm control is found, RunSpeech is called.
- (3) Else, the QA control is run, with the selected confirm control as the active question control.

10

15

20

For multi-modal browsers, only the grammar loading and event dispatching steps are carried out.